DTIC FILE COPY

AD-A229 822

AAMRL-TR-90-042

NEURAL NETWORK BASED HUMAN PERFORMANCE MODELING (U)

EDWARD L. FIX

AUGUST 1990

FINAL REPORT FOR PERIOD FROM SEPTEMBER 1988 to SEPTEMBER 1990

Approved for public release; distribution is unlimited

DTIC
ELECTE
DEC 0 3 1990
S B D
Co

ARMSTRONG AEROSPACE MEDICAL RESEARCH LABORATORY
HUMAN SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-6573

90

NOTICES

When US Government drawings, specifications, or other data are used for any purpose other than a definitely related Government procurement operation, the Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

Please do not request copies of this report from the Armstrong Aerospace Medical Research Laboratory. Additional copies may be purchased from:

National Technical Information Service
5285 Port Royal Road
Springfield, Virginia 22161

Federal Government agencies and their contractors registered with the Defense Technical Information Center should direct requests for copies of this report to:

Defense Technical Information Center
Cameron Station
Alexandria, Virginia 22314

TECHNICAL REVIEW AND APPROVAL

AAMRL-TR-90-042

This report has been reviewed by the Office of Public Affairs (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

The voluntary informed consent of the subjects used in this research was obtained as required by Air Force Regulation 169-3.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

CHARLES BATES, JR.
Director, Human Engineering Division
Armstrong Aerospace Medical Research Laboratory

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 1990	Final, 1989 - 1990

4. TITLE AND SUBTITLE

Neural Network Based Human Performance Modeling (U)

5. FUNDING NUMBERS

62202F 6893 04 69

F33615-89-C-0532

6. AUTHOR(S)

Edward L. Fix, Major

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Armstrong Aerospace Medical Research Laboratory
Wright-Patterson AFB OH 45433-6573

8. PERFORMING ORGANIZATION REPORT NUMBER

AAMRL-TR-90-042

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

A

13. ABSTRACT *(Maximum 200 words)*

Neural networks provide an alternative method of building models of human performance. They can learn behavior from examples, reducing the need for many identical repetitions and intensive analysis. A properly trained net can be very robust in its response to a novel stimulus. This opens the door to modeling performance in the presence of an interactive stimulus. Neural networks provide the possibility of robust models that can operate interactively in real time, depending on the size and architecture of the net and the application.

A neural network architecture derived from recurrent back propagation is presented which learns to mimic human behavior and performance in a sample task. It shows operating characteristics similar to those of human subjects, and even makes the same kinds of mistakes. Possible applications are discussed.

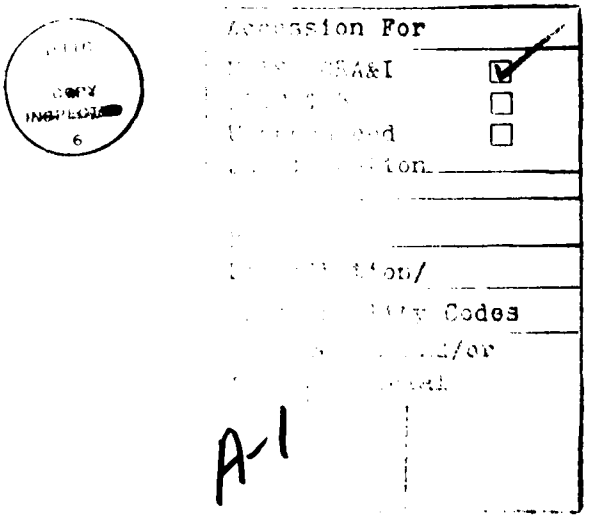
14. SUBJECT TERMS		15. NUMBER OF PAGES
Human factors, Neural Networks, Modeling	Human Performance, Artificial Intelligence, (JT)	138
		16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Summary

Neural networks provide an alternative method of building models of human performance. They can learn behavior from examples, reducing the need for many identical repetitions and intensive analysis. A properly trained net can be very robust in its response to a novel stimulus. This opens the door to modeling performance in the presence of an interactive stimulus. Neural networks provide the possibility of robust models that can operate interactively in real time, depending on the size and architecture of the net and the application.

A neural network architecture derived from recurrent back propagation is presented which learns to mimic human behavior and performance in a sample task. It shows operating characteristics similar to those of human subjects, and even makes the same kinds of mistakes. Possible applications are discussed.



Preface

This report documents an experiment performed by the Harry G. Armstrong Aerospace Medical Research Laboratory (AAMRL), Human Engineering Division, Crew Station Integration Branch, Manned Threat Quantification (MTQ) Program as part of Program Element 62202F and Project/Task/Work Unit 6893/04/69. Operations and maintenance support was provided by Logicon Technical Services, Inc., Dayton, Ohio, under Contract F 33615-89-C-0532. The principal investigator for this experiment was Maj Edward Fix. The author wishes to acknowledge the many contributions of the following people:

- LtCol William Marshak for guidance and support
- Mr. Chuck Goodyear for statistical analysis and suggestions
- Dr. Barry Deer for ideas
- All the subjects who volunteered a couple of hours of their time.

Contents

1 Introduction 1

2 Experimental Task 3

2.1 Performance 3

2.2 Data Collection 5

2.3 Analysis 5

3 Neural Network Models 8

3.1 Single Node Perceptrons 8

3.2 Multi-Layer Perceptrons 10

3.3 Perceptron Simulation 11

3.4 Input and Output Representations 11

3.5 Architecture 13

3.6 Training 14

4 Results 16

5 Conclusion 19

5.1 Lessons Learned 19

5.2 Future Work 20

A Model Results Plots 21

B Simulation C Source Code 49

C Network C Source Code 77

D Net Testing C Source Code 101

References 129

List of Figures

2.1 Experimental Task Screen 4
2.2 Distance Behind at Lane Change (D_L) 6
2.3 Closest Approach in Same Lane (D_B) 6
2.4 Distance Between at Lane Change (D_C) 7

3.1 Single Perceptron Node 9
3.2 Sigmoid Function 9
3.3 Multi-Layer Feed-Forward Network 10
3.4 Network Input Representation 12
3.5 Network Architecture 14

A.1 Sub. 1 Speed Frequency and Data Points 22
A.2 Sub. 1 Relative Frequency of Data 23
A.3 Sub. 2 Speed Frequency and Data Points 24
A.4 Sub. 2 Relative Frequency of Data 25
A.5 Sub. 3 Speed Frequency and Data Points 26
A.6 Sub. 3 Relative Frequency of Data 27
A.7 Sub. 4 Speed Frequency and Data Points 28
A.8 Sub. 4 Relative Frequency of Data 29
A.9 Sub. 5 Speed Frequency and Data Points 30
A.10 Sub. 5 Relative Frequency of Data 31
A.11 Sub. 6 Speed Frequency and Data Points 32
A.12 Sub. 6 Relative Frequency of Data 33
A.13 Sub. 7 Speed Frequency and Data Points 34
A.14 Sub. 7 Relative Frequency of Data 35
A.15 Sub. 8 Speed Frequency and Data Points 36
A.16 Sub. 8 Relative Frequency of Data 37
A.17 Sub. 9 Speed Frequency and Data Points 38

A.18 Sub. 9 Relative Frequency of Data 39
A.19 Sub. 10 Speed Frequency and Data Points 40
A.20 Sub. 10 Relative Frequency of Data 41
A.21 Sub. 11 Speed Frequency and Data Points 42
A.22 Sub. 11 Relative Frequency of Data 43
A.23 Sub. 12 Speed Frequency and Data Points 44
A.24 Sub. 12 Relative Frequency of Data 45
A.25 Sub. 13 Speed Frequency and Data Points 46
A.26 Sub. 13 Relative Frequency of Data 47

List of Tables

4.1 Variable Cars Model Match 17
4.2 Hostile Cars Model Match 18

Introduction

There are many techniques for modeling systems and operations, including human operated systems. The Manned Threat Quantification (MTQ) program at AAMRL has developed models of human operation of air defense systems using a variety of techniques. These models have been used for purposes including systems analysis and design, countermeasures planning, etc. A common thread of these models is that they are based on data gathered from realistic, man-in-the-loop simulations in the laboratory. This has made them very useful to the users. However, they have some weaknesses.

The models are difficult and expensive to develop. It is necessary to recreate a realistic operating position in the laboratory. The statistical nature of the analysis requires multiple repetitions of each situation, and careful selection of the situations to present. The multiple repetition requirement means the stimulus (i.e. the aircraft that the air defense crew is trying to shoot down) must not react to the subject crew's actions. Therefore, while the model may predict the vulnerability of the target aircraft, it does not model the air defense crew's response to any novel aircraft action. If the aircrew reacts to the defense crew's action and deviates from the flight path that was used to develop the statistical base of the model, that model starts becoming invalid to some degree. Therefore, it is not well suited to real-time simulation against a live aircrew.

The purpose of this effort is to develop a method for modeling the actions of a person or crew that will react correctly to novel situations. It must be able to use data from an uncontrolled source where the situations never repeat (that is, an interactive, realistic environment) and traditional statistical analysis is not useful. The model could then be based on a broad range of situations and may be more robust than those made using other modeling techniques.

This experiment used the emerging technology of neural networks as a modeling technique. Neural networks are computer programs based loosely on what is known about the architecture of the brain. They are "trained" on examples of behavior, and "learn" the

correct responses. Repetitive stimuli are not required, or even desirable. The net can learn appropriate responses from data gathered from a realistic environment. Extensive analysis is not required; it is not necessary to know the exact reasons for a particular response, only to show the network the input parameters, and train it to the correct response. Neural network performance degrades gracefully in the presence of uncertain or noisy data. By nature, they give uncertain or best guess responses when the input data is uncertain. When presented with novel stimuli, they give answers interpolated or extrapolated from training examples.

A simple video simulation like that described by Shepanski and Macy, was adapted from an implementation by Restrepo as a test for the neural network modeling technique. Neural network architectures and training techniques were explored and extended to achieve this unique application of the technology.

Experimental Task

The task the subjects performed was based on a popular neural network demonstration concept (Shepanski and Macy, Restrepo). It is a computer generated display showing a two-lane circular track with several "cars" (Fig 2.1). One car is controlled by the subject, and the others are controlled by the computer. The cars all travel in a counterclockwise direction, but the perspective is adjusted so that the subject's car is always at the 3 o'clock position on the track, and everything else moves relative to the controlled car.

The subject's task was to drive his car around the track, switching lanes and adjusting speed as necessary to avoid collisions. Although the subject was not instructed in specific driving goals, there was a score presentation that incremented one point each time the subject passed another car, and decremented three whenever the subject bumped another car. The instructions were vague in order to elicit different driving techniques from different subjects.

The subject controlled his car through a mouse interface. The mouse pointer moved around on a "control panel". Putting the pointer in the left half of the panel put the car in the left (inside) lane, and the right half put it into the right lane. Moving the mouse to the top of the panel accelerated the car to full speed, and moving it to the bottom stopped the car.

2.1 Performance

There were two different testing scenarios, with two levels of interactivity in each. In one scenario, there were four computer controlled cars. They started at random positions on the track, two in each lane, traveling at random speeds. After a practice period, data was gathered for four minutes. This scenario is referred to as the "continuous data" case.

In the second scenario, there were only two computer controlled cars, and they started at the 9 o'clock position of the track (opposite the subject's car). They were either together

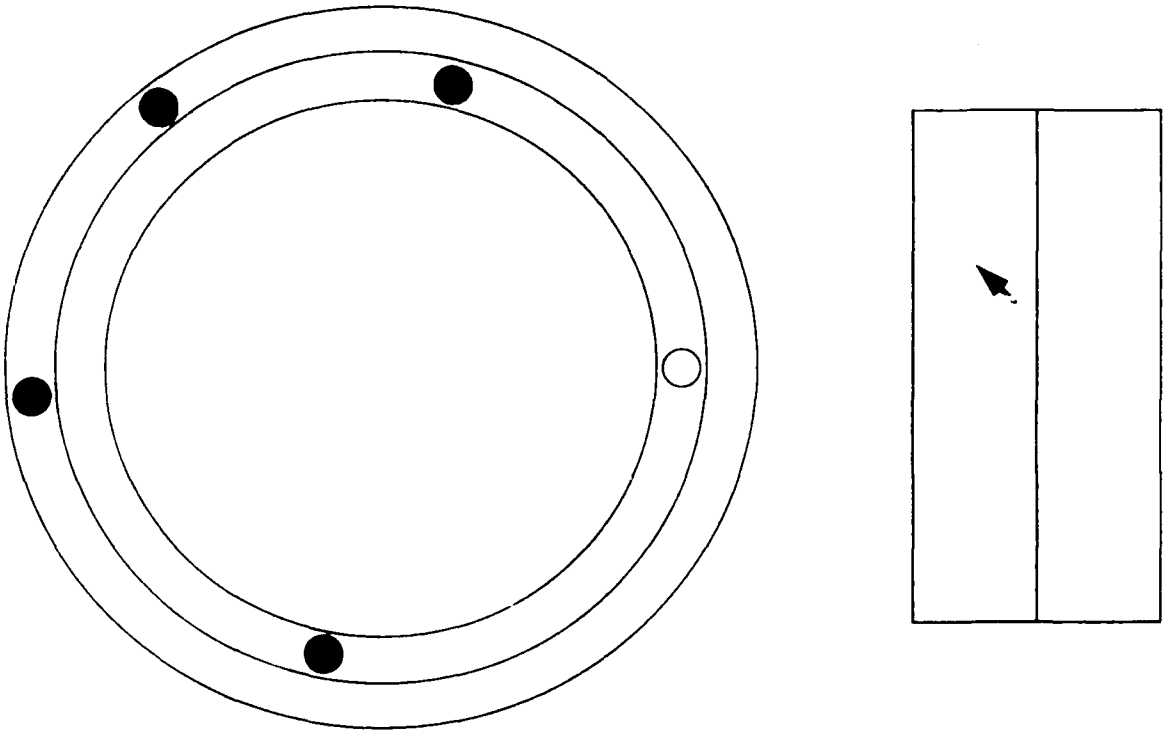


Figure 2.1: Experimental Task Screen

or spaced somewhat apart, and were set up with speeds that would either separate them or close them together at a slow or fast rate. There were 14 different setup conditions altogether, and each was repeated eight times in randomized order. Each setup first came on with the screen frozen to allow the subject to get a feel for the scenario. The subject started the simulation, and it continued until the subject had passed both of the other cars. This scenario is referred to the "setup data" case.

The experiment included two levels of interactivity. The first was called "Variable Cars". At infrequent, random times the computer controlled cars changed speed, and if one car approached another from behind, it switched lanes to pass. This was at least indirectly interactive because the exact situation the subject faced on the track was related to the speed with which he approached the car or group of cars ahead. In the setup data case, the cars did not vary speed or lane as they did in the continuous data case.

In the second level of interactivity, called "Hostile Cars", the computer controlled cars changed speeds and lanes as in the first, but in addition they actively tried to prevent the subject from passing. They sped up as the subject approached, and switched lanes, or matched speeds with a nearby car in the other lane to block the subject. The setup data

case of hostile cars was the same as that for the variable cars, with the addition that the cars tried to prevent the subject from passing as in the continuous data case.

Driver training for a subject was accomplished by exposing the subject to increasing levels of difficulty. The subject first ran a version where the other cars never changed speed or lane until he felt comfortable controlling the simulation. When the subject was consistently passing the other cars without collisions, the demonstration program was stopped and the "Variable Cars" program started. The subject was given as much time as necessary to become accustomed to the new situation, and then data were gathered for the continuous case. At the completion of a four minute data collection run, the setup conditions for the variable cars case were presented. The familiarization, continuous data, and setup data sequence was repeated for the "Hostile Cars" case.

2.2 Data Collection

While the subject was performing the task, the position, lane, and speed of each car, including the subject's car, were recorded each time the screen animation was updated. After a net was trained, it was tested by presenting a novel starting position and letting it control the simulation. Its output was gathered in the same way.

2.3 Analysis

The data from both the subject and net were analyzed for operating style. Parameters gathered included the distance the subject or net was behind the nearest car in the same lane when it switched lanes to pass (D_L) (Fig 2.2), the closest approach to the nearest car ahead in the same lane when the controlled car slowed enough to allow the computer car to pull further ahead (D_B) (Fig 2.3), the distance between cars that were close ahead and close behind in opposite lanes when the controlled car switched lanes to pass between them (D_C) (Fig 2.4), and number of cars passed with number of collisions. These can be compared for similarities and differences.

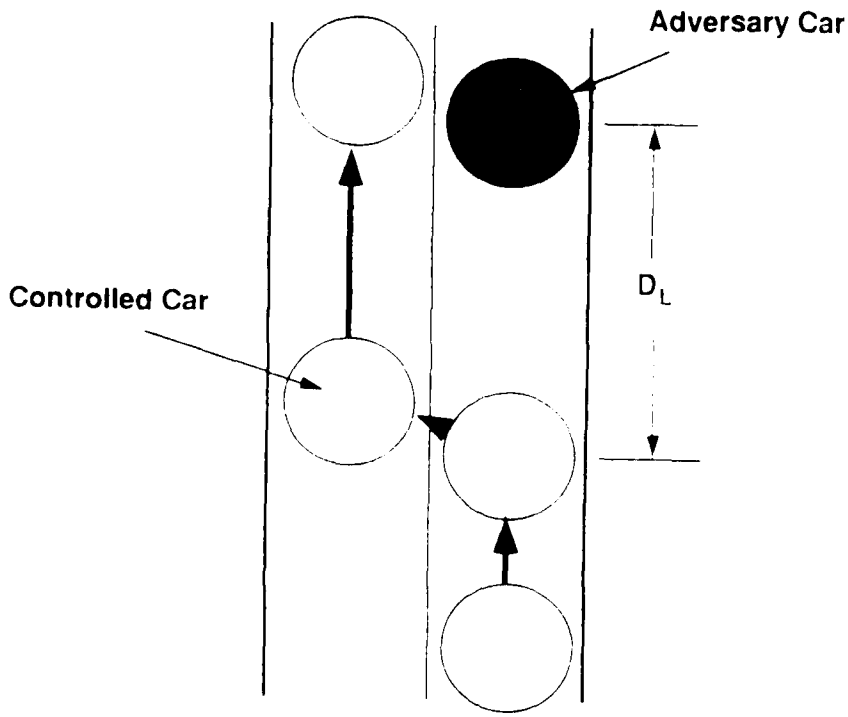


Figure 2.2: Distance Behind at Lane Change (D_L)

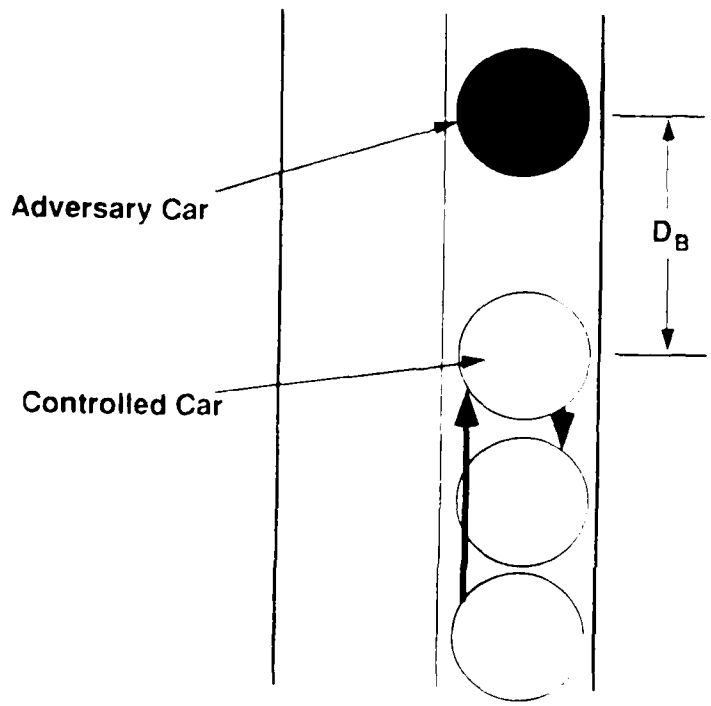


Figure 2.3: Closest Approach in Same Lane (D_B)

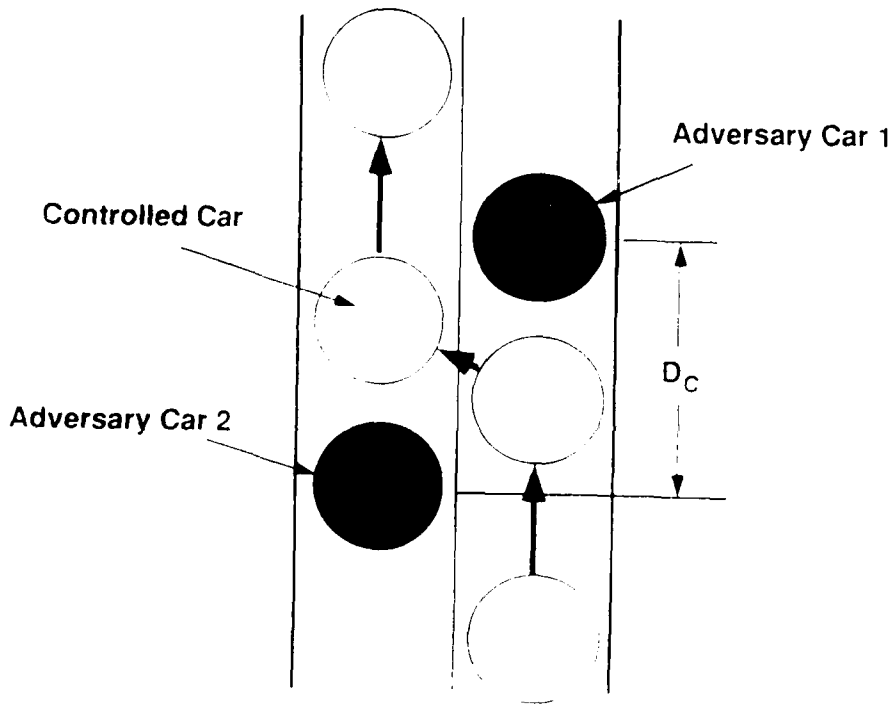


Figure 2.4: Distance Between at Lane Change (D_C)

Neural Network Models

Neural networks "attempt to achieve good performance via dense interconnection of simple computational elements. In this respect, artificial neural net structure is based on our present understanding of biological nervous systems" (Lippmann). A type of neural net known as a multi-layer perceptron seemed to be a logical approach.

3.1 Single Node Perceptrons

A single computational element or neuromime is shown in Fig 3.1. The output value is given by is the sigmoid equation (Fig 3.2) and x represents an input vector element, w represents the

$$y = f(\sum_{i=1}^{N} w_i x_i - \theta) \tag{3.1}$$

where

$$f(\alpha) = \frac{1}{1 + e^{-\alpha}} \tag{3.2}$$

connection weight, and θ is a small random threshold. N is the number of elements in the input vector. It can be shown (Lippmann) that Eq 3.1 describes a hyperplane boundary (a straight line if there are two inputs) in N-dimensional space between two regions. If vectors $\mathbf{x} = \{x_1, \ldots, x_N\}$ which are separable into two regions are applied to the inputs, the weights can be adapted so that the hyperplane divides the two regions of points. The training algorithm is

$$\Delta w_i = \eta(d - y)x_i \tag{3.3}$$

$$1 \leq i \leq N$$

$$0 < \eta < 1$$

where d is the desired output (0 or 1). After a number of training trials, the perceptron may converge to a solution. In this way, the perceptron can classify the input vectors. The

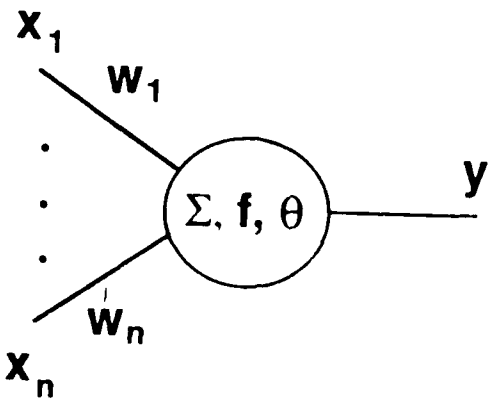


Figure 3.1: Single Perceptron Node

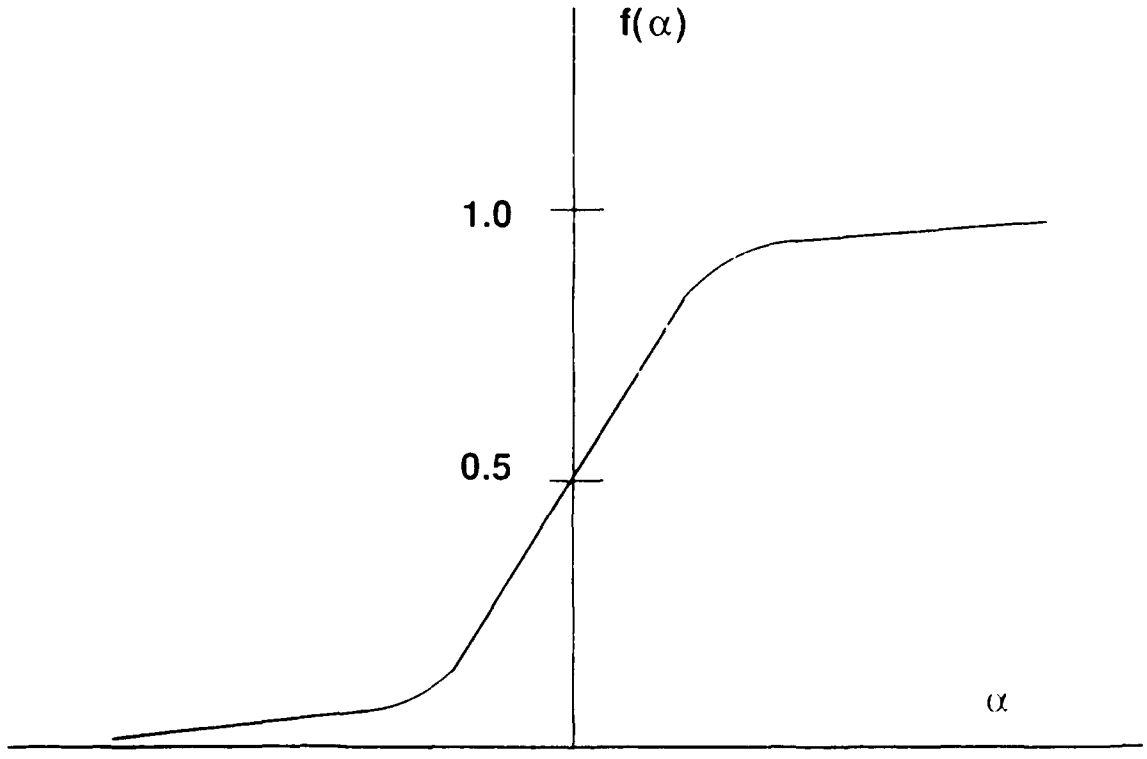


Figure 3.2: Sigmoid Function

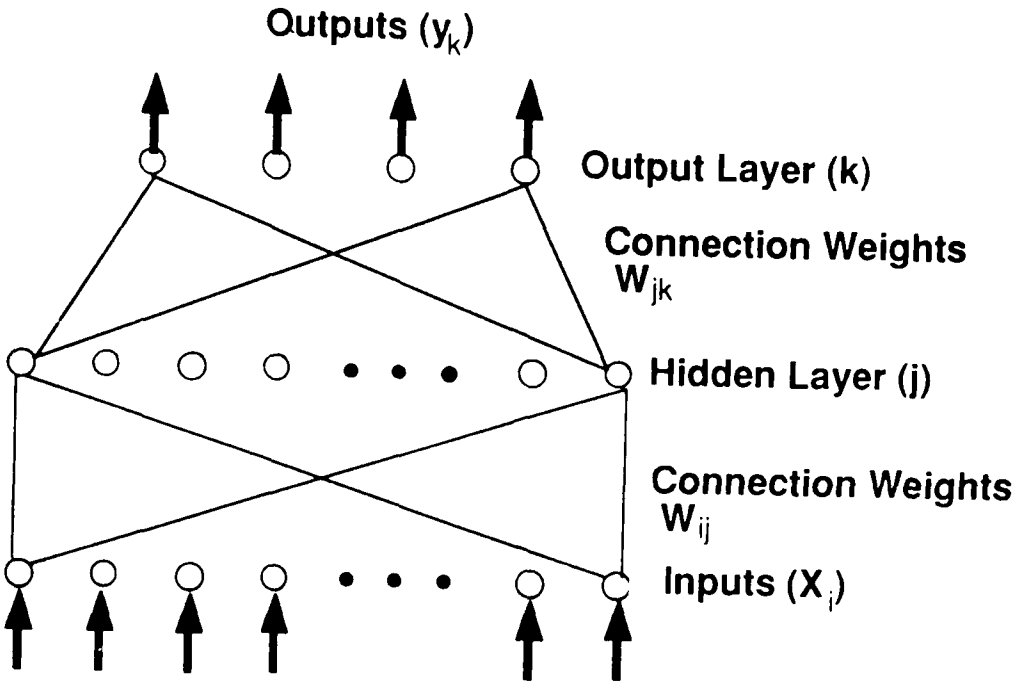


Figure 3.3: Multi-Layer Feed-Forward Network

output can also be trained to continuous values from 0 to 1, to approximate continuous functions.

3.2 Multi-Layer Perceptrons

It can be shown that an arrangement of several nodes in each of three layers, where all nodes in one layer (or all inputs) are connected to all nodes of the next layer, can separate an arbitrary number of classes and regions with arbitrarily complex boundaries. This arrangement is schematically shown in Figure 3.3. The complexity a given arrangement can handle depends on the number of nodes in each layer.

The extended training algorithm is called back propagation:

$$\Delta w_{bc} = \eta \delta_c y_b \tag{3.4}$$

where

$$\delta_c = y_c(1 - y_c)(d_c - y_c) \tag{3.5}$$

if the current layer is the output where d_c is the desired output of node c and y_c is the

actual output or

$$\delta_c = y_c(1 - y_c) \sum_a \delta_a w_{ca} \tag{3.6}$$

if the current layer is an inner or hidden layer. In Eqs 3.4 through 3.6, x denotes an input to a node and y is its output. Note that the output on one node can be the input to another in the next layer. The subscript c denotes the current layer, while a denotes the layer above and b denotes the layer below. The θ values in Eq. 3.1 are also adapted by back propagation. A more complete description and derivation can be found in Rumelhart, Hinton and Williams.

3.3 Perceptron Simulation

Although perceptrons are conceptually implemented as massively parallel networks of simple processors, they can be simulated on a conventional digital computer. These simulations are very computation intensive, but if the net is small enough, it may be possible to run the simulation in real time as a subroutine or on an appropriate external processor. The back propagation training algorithm is the most time consuming part, but once the net is trained the weights can be transferred to a real time processor.

3.4 Input and Output Representations

The decision was made that the representations to the net should match what the subject saw as closely as possible. For that reason, the data from the computer controlled cars were presented to the network sorted by position. That is, input no. 1 is the nearest car in the left lane, no. 2 is the nearest in the right lane, and nos. 3 and 4 are the farther cars in the left and right lanes respectively. The parameter input to the network was the angle of each car; A number between 0 and 1, with the 0 point at the position of the subject's car, and increasing counterclockwise.

To emulate a memory of recent motion, from which speeds and changing relationships could be derived, the positions of the computer cars were presented for the latest 10 cycles. Each cycle represents a screen animation update. At each new time cycle, the earliest position is dropped off, all the remaining 36 inputs (9 cycles of 4 inputs) are shifted, and the current positions are added at the latest time position. This provided the net with the

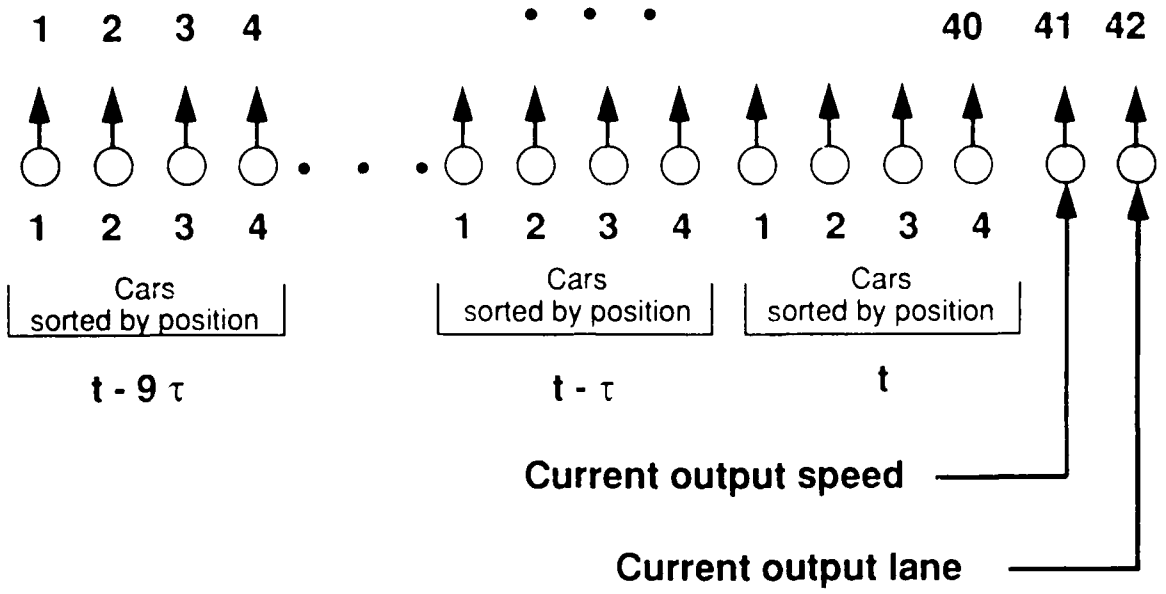


Figure 3.4: Network Input Representation

information to "judge" relative and absolute speeds in much the same way that a human would.

In the continuous data case with four other cars, this makes 40 inputs to the net (Fig 3.4), and the two-car setups give 20 inputs. In both cases, there are two more inputs; the current speed normalized to the interval [0,1] and the current lane, either 0 or 1. In summary, there were 42 inputs to the nets for the continuous data cases, and 22 for the setup conditions.

In all cases, the networks had two outputs. One output was used to represent speed and the other to represent lane choice. This is unusual in that one output is used to represent a continuously variable value (speed) and the other represents a discrete (0 or 1) value. This architecture is somewhat unique in that most network implementations use either discrete or continuously variable values but not both.

The speed of each car was represented internally in the simulation on an arbitrary scale from 0—15. The speed output of the network is trained on a value of 0—1 where 1 represents a value of 20. This is to allow the network to more easily attain the maximum speed. The actual speed value a trained network feeds to the simulation, however, is capped at 15. The lane output is represented to the simulation as 0 if the actual output is less than 0.5 and as 1 if the output is greater than 0.5. There is an additional constraint that the lane may not be changed unless the output of the network is more than 0.55 different from the previous

lane value. That is, if the previous lane value was 0, the network output must be greater than 0.55 for the lane value to switch to 1, and if the previous value was 1, the output must be less than 0.45 for the lane to switch to 0. This prevents "jitter" if the net is uncertain and the value is hovering around the midrange.

3.5 Architecture

The overall architecture was based on the standard feed-forward network architecture described in section 3.2. One hidden layer was used, containing 20 nodes, and there were two output nodes. One output was used to represent the speed setting, and the other for the lane choice.

Two major modifications were made to this standard architecture. First, recursion was used in the hidden layer to introduce a time delay to emulate reaction time. Originally, this was accomplished by treating the outputs of the hidden units from the previous iteration as additional inputs, fully connected to the hidden layer (Pineda, Almeida). Training was accomplished by straightforward application of the back propagation algorithm to the recursive weights as well as the feed forward weights. However, examination of the trained weights revealed that the weight connecting each hidden node to itself was at least an order of magnitude larger than the weights connecting that node to the other hidden nodes. Therefore, the architecture was modified to include only one connection from the output of each hidden unit to its own input. This architecture and training caused the net to behave with realistic time delays, and sometimes a rapid change would overshoot the target value, just as would be observed in human performance.

The second modification to the net architecture was necessary because the net failed to recognize and treat rare occurrences adequately. It was discovered that subjects tended to act quite differently in two different circumstances. In the case where the other cars were spaced out ahead of the subject's car, the driver tended to drive full speed and merely switch lanes to avoid collisions. This occurred most of the time. However, when there were cars in both lanes ahead of the subject's car, with insufficient room to pass between them, the subject would slow down to their speed or less and wait for them to separate. The subject would sometimes abruptly slow too much and overshoot the desired slower speed. This was especially true in the "Hostile Cars" case. The net, trained on the whole subject

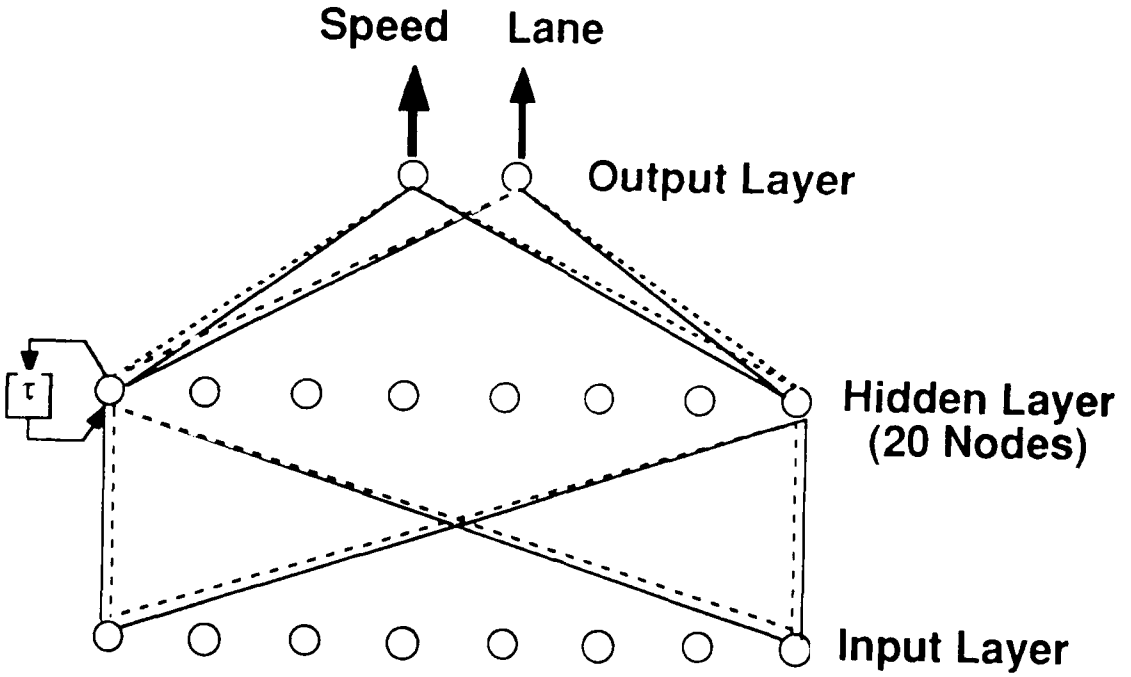


Figure 3.5: Network Architecture

data set, was unable to emulate this dual strategy characteristic.

The solution was to use two redundant sets of feed forward weights; in effect use two nets with a common recursive connection (Fig 3.5). One set of weights emulates behavior in the frequent condition and the other in the rare condition. This architecture is essentially a neural network set controlled by a simple expert system. Arrangements of this type have been described before (Holden), and appear to combine the strengths of neural and expert systems.

3.6 Training

The back-propagation training algorithm was modified as described by Fix (1988). Briefly, the $y(1-y)$ term in Eq 3.5 at the output layer is replaced by

$$y(1-y)\,\mathrm{u}(flip) + 0.25\,\mathrm{u}(-flip) \tag{3.7}$$

where:

$$flip = \mathrm{sgn}(d - 1/2)(y - 1/2) \tag{3.8}$$

$\mathrm{Sgn}(x)$ is the signum function that returns +1 if the argument is positive or -1 if the argument is negative, and $u(x)$ is the unit step function that returns 0 for negative arguments and 1 for positive arguments. This modification can speed training in the initial stages.

The networks were trained using the data gathered from the subjects. Both redundant sets of weights, the recursive weights, and the thresholds were initialized to small random values between -1 and 1. The hidden layer outputs were initialized to 0.0 for the recursion, and all time slices in the input layer were loaded to the initial positions of the cars. Training was incremental – that is, the weights were updated after each presentation of an input vector. The training vectors were always presented in the order in which they occurred in the training data set. For each subject, four nets were trained, one each for variable cars continuous data and setup data, and for hostile cars continuous and setup data.

Training progress was measured by a mean absolute error value. This was calculated by computing the absolute value of the error at each output node after presenting an input vector and averaging that value over the output nodes. That was repeated for each input/output training vector pair, and averaged over all. At the completion of each cycle through the training data set, this value was recorded and reset for the next cycle. When the rate of change of the error value became small, the net was considered trained. The final error value of the networks was typically between 0.002 and 0.006.

Results

Only results from the continuous data cases are presented here. The network models were compared to the human subjects by four measures of operating style besides the number of cars passes versus number of collisions. The four parameters were:

D_L– Distance behind nearest adversary car at lane change.

D_C– Distance between two adversary cars in opposite lanes when passing between them.

D_B– Closest approach to nearest car in same lane when slowing to back off to avoid a collision.

Speed– Relative frequency plot of speed.

These parameters were gathered for both the subject data used to train the network, and from the network model when it was tested after training. These measures were compared between each subject and the model trained from that subject for both the variable cars and hostile cars scenarios.

Because of the large variability in the conditions, and the fact that no criteria for "good enough" model fit were established, it was decided to present the data visually rather than perform statistical comparisons. The plots of these data are given in Appendix A.

The first page for each subject shows relative frequency plots of speed for both the variable and hostile cases, with the number of data points in D_L, D_C, and D_B. Since most subjects spent a disproportionately large amount of time at full speed, the number of data points at full speed (15) is given separately from the plot. This allows more detail to be observed in the rest of the plot.

The second page contains plots showing the distribution of D_L, D_C, and D_B. The large circle in each plot represents the track in the original simulation. It is divided into segments, each containing a small, solid circle. The area of each solid circle represents the relative frequency of occurrence of values in that segment. From thi· the observer can get a sense of the comparative distribution of each parameter between the subject and network model.

Match Between Subject and Network					
Subject	Speed	D_D	D_C	D_L	Match
1	-	-	+	+	-
2	+	-	-	+	-
3	+	-	+	+	+
4	+	-	+	+	+
5	+	0	-	0	0
6	0	-	-	+	0
7	-	-	+	+	-
8	+	+	+	+	+
9	-	+	+	+	+
10	+	+	+	-	+
11	-	-	+	+	-
12	-	-	-	+	-
13	+	-	+	-	-
+ = Good, - = Fair, 0 = Poor					

Table 4.1: Variable Cars Model Match

Tables 4.1 and 4.2 give subjective estimations of the success of each model in emulating the respective subject. In the variable cars case, there were 5 good matches, 6 fair matches, and 2 that did not match. In the hostile case, there were 4 good matches, 2 fair matches, and 7 that did not match. This assessment is offered only as a top-level comparison between paradigms, and as an indication that this technique shows promise as a modeling technique. No objective criteria were established for goodness of fit, and the observer is free to apply his own criteria in evaluating the results. However, the overall match was judged good if no more than one single parameter was fair, poor if any single parameter was poor, and fair otherwise.

Once a network architecture and training paradigm were found to work acceptably on one subject, the same were applied to all the subjects. As was observed on the original subject, things like gathering another training data set, changing the input representation, changing the number of hidden nodes, and in general tinkering with the network can produce improved results. Therefore, it is not particularly discouraging that there were some failures with the architecture that was used.

Match Between Subject and Network					
Subject	Speed	D_H	D_C	D_L	Match
1	-	+	+	+	+
2	0	-	0	0	0
3	+	-	0	-	0
4	0	+	-	-	0
5	0	0	-	0	0
6	0	0	-	0	0
7	+	-	+	+	+
8	+	+	+	+	+
9	-	+	+	+	+
10	-	-	-	-	-
11	-	-	-	0	0
12	-	-	-	0	0
13	+	-	+	+	+
+ = Good, - = Fair, 0 = Poor					

Table 4.2: Hostile Cars Model Match

Conclusion

A neural network architecture has been developed which shows promise for modeling and emulating human behavior and performance. It is based on a multi-layer, feed-forward architecture, but has a more complex architecture.

The hidden layer has recursive connections which allow the network to emulate reaction time. The architecture also includes multiple sets of feed-forward connection weights. These different weights are trained and used under different situations to emulate different strategies. This makes the overall system a hybrid neural network/expert system.

The system has successfully emulated human performance in a computer simulation of a driving task. It will be adapted to more complex and realistic tasks in the near future.

5.1 Lessons Learned

It is important that the input to the network matches what the human operator sees as closely as possible. This allows the network to interpolate and extrapolate from the training examples more reliably. The internal transforms the net is using may be closer to what the person used.

To successfully emulate human behavior, it is necessary to build in reaction time. This was done here with the recursive weights in the hidden layer. This means the network will not be able to change instantaneously to changing circumstances, but must change smoothly.

Each unique situation that elicits a different strategy from the operator must be represented by a separate set of feed-forward connection weights. The resulting expert system "executive" can be very simple. It only looks at the external situation and switches in the correct network to give the correct actions.

Highly skilled operators are easier to model then less skilled ones. They tend to have consistent strategies which the network is able to emulate. A subject who is more erratic

tends to "confuse" the net and it does not converge as well. One possible way to avoid this problem is to carefully pick the examples from the training set to actually train the network. This will allow the network to train on the provided samples, but care must be taken to ensure the chosen samples are truly representative of the desired behavior.

5.2 Future Work

First, we will continue to adapt this modeling technique to the setup data cases. This data presents unique challenges and opportunities due to the limited engagement time of each setup, and the multiplicity of runs and repeatability of the environment.

We will extend the technique to the performance of the crew of a surface-to-air (SAM) missile system. This is being done at AAMRL in the Threat Assessment Facility by linking an existing simulation of a Soviet SAM with a simulation of a B-52 electronic warfare station. These will be run in an interactive, two-sided simulation to gather data for training the model.

Once a set of tools has been built that can model a SAM system, we will get data from some of the many realistic training and simulation facilities. These could include the training ranges at Navy Fallon or Nellis, or the REDCAP facility. We will then be able to use this data to build high fidelity models.

Models generated from these realistic environments could be used to run real-time simulations to give aircrews intelligent adversaries. Two obvious applications are unmanned threat emitter sites on the training ranges or SAC low level training routes, and aircrew training simulators including weapon system trainers.

Appendix A

Model Results Plots

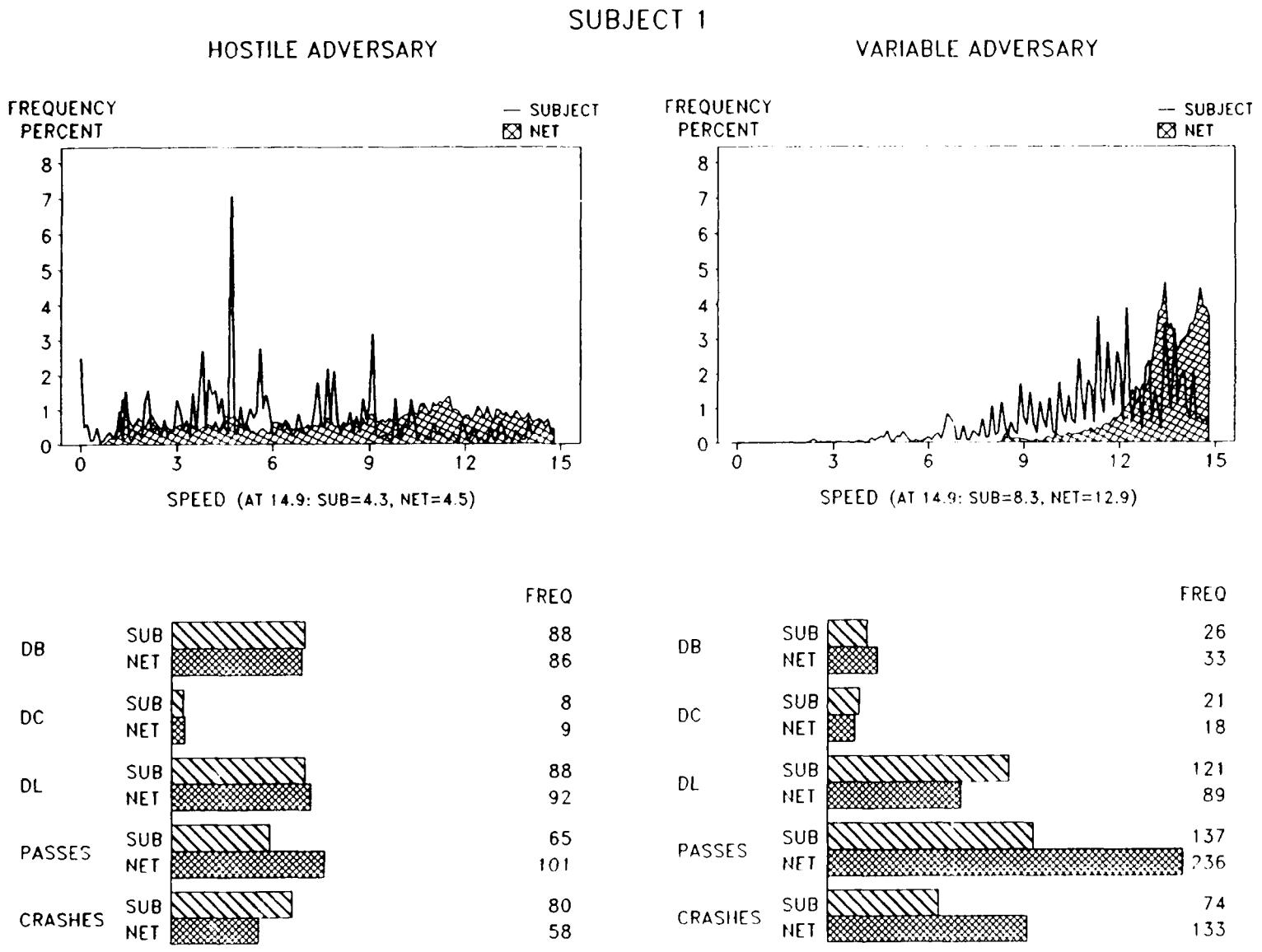


Figure A.1: Sub. 1 Speed Frequency and Data Points

Figure A.2: Sub. 1 Relative Frequency of Data

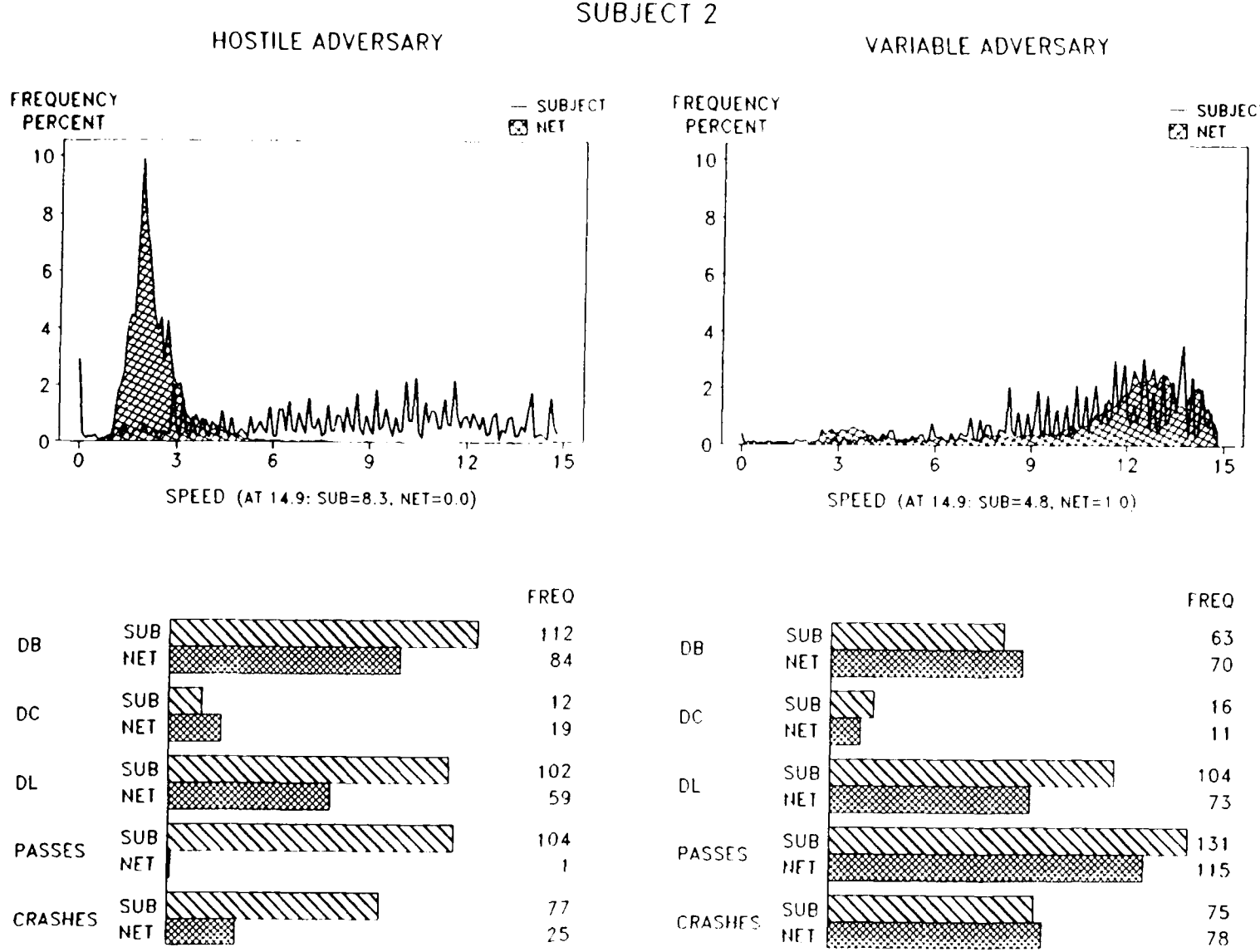


Figure A.3: Sub. 2 Speed Frequency and Data Points

Figure A.4: Sub. 2 Relative Frequency of Data

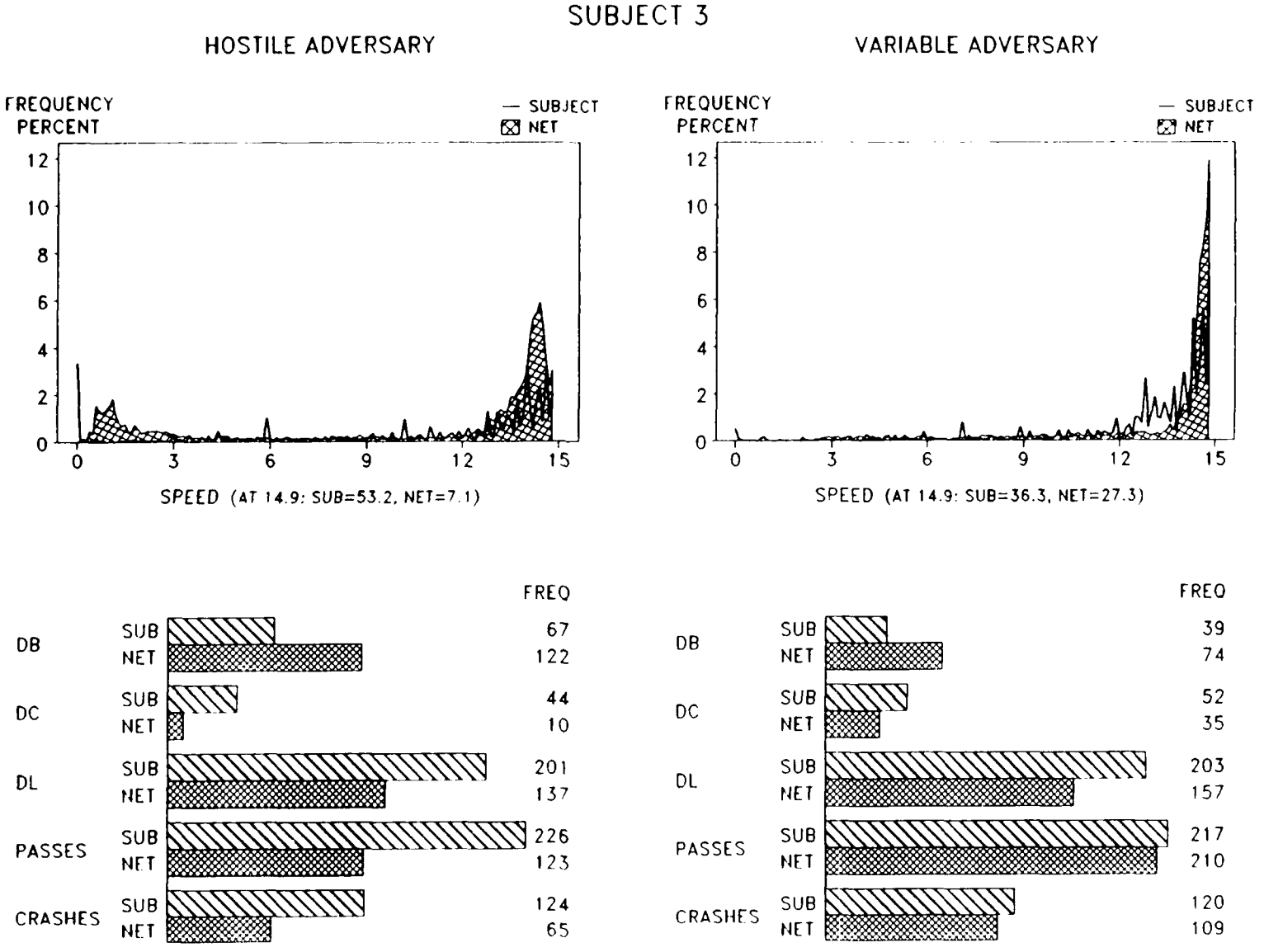


Figure A.5: Sub. 3 Speed Frequency and Data Points

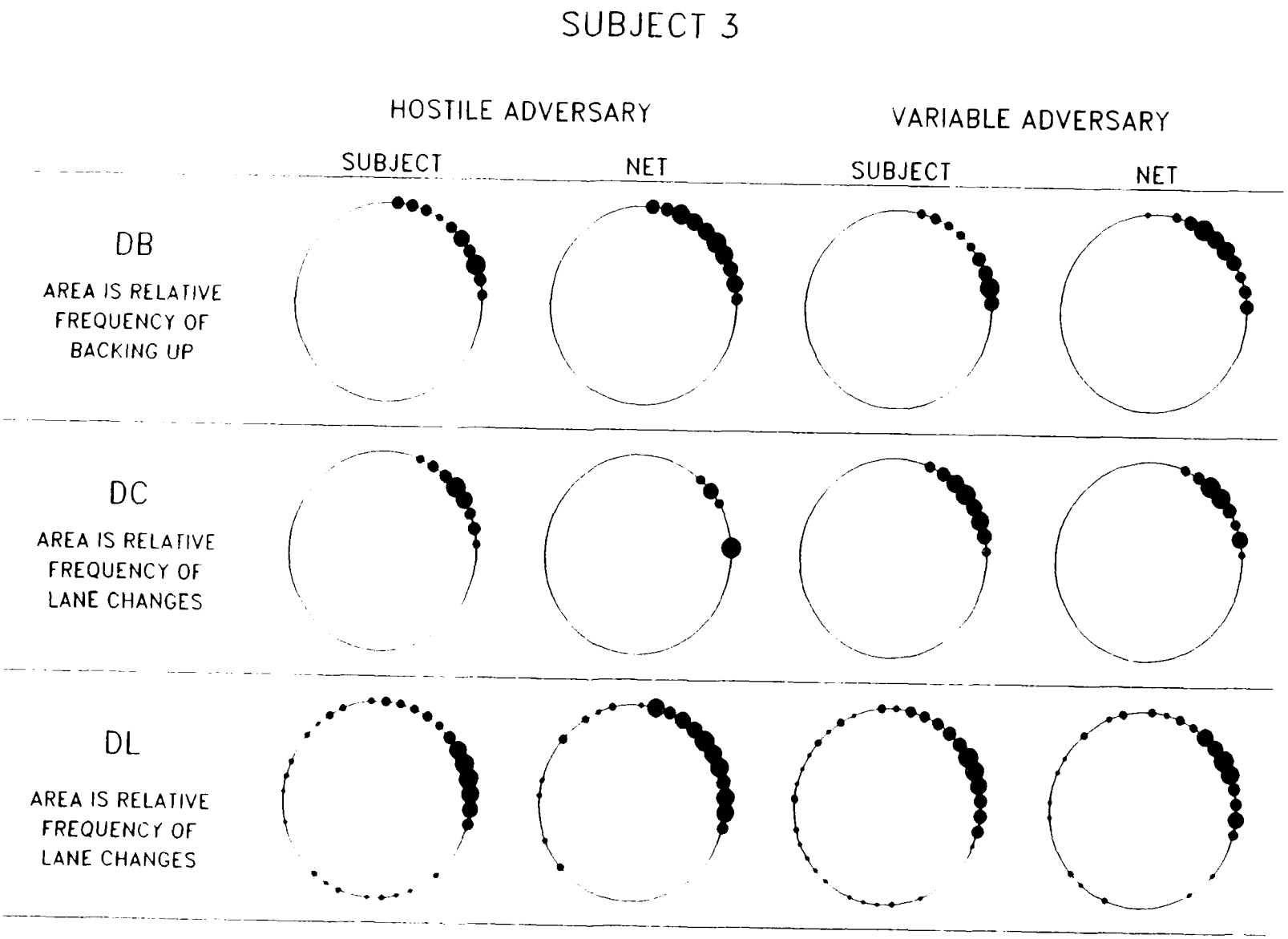


Figure A.6: Sub. 3 Relative Frequency of Data

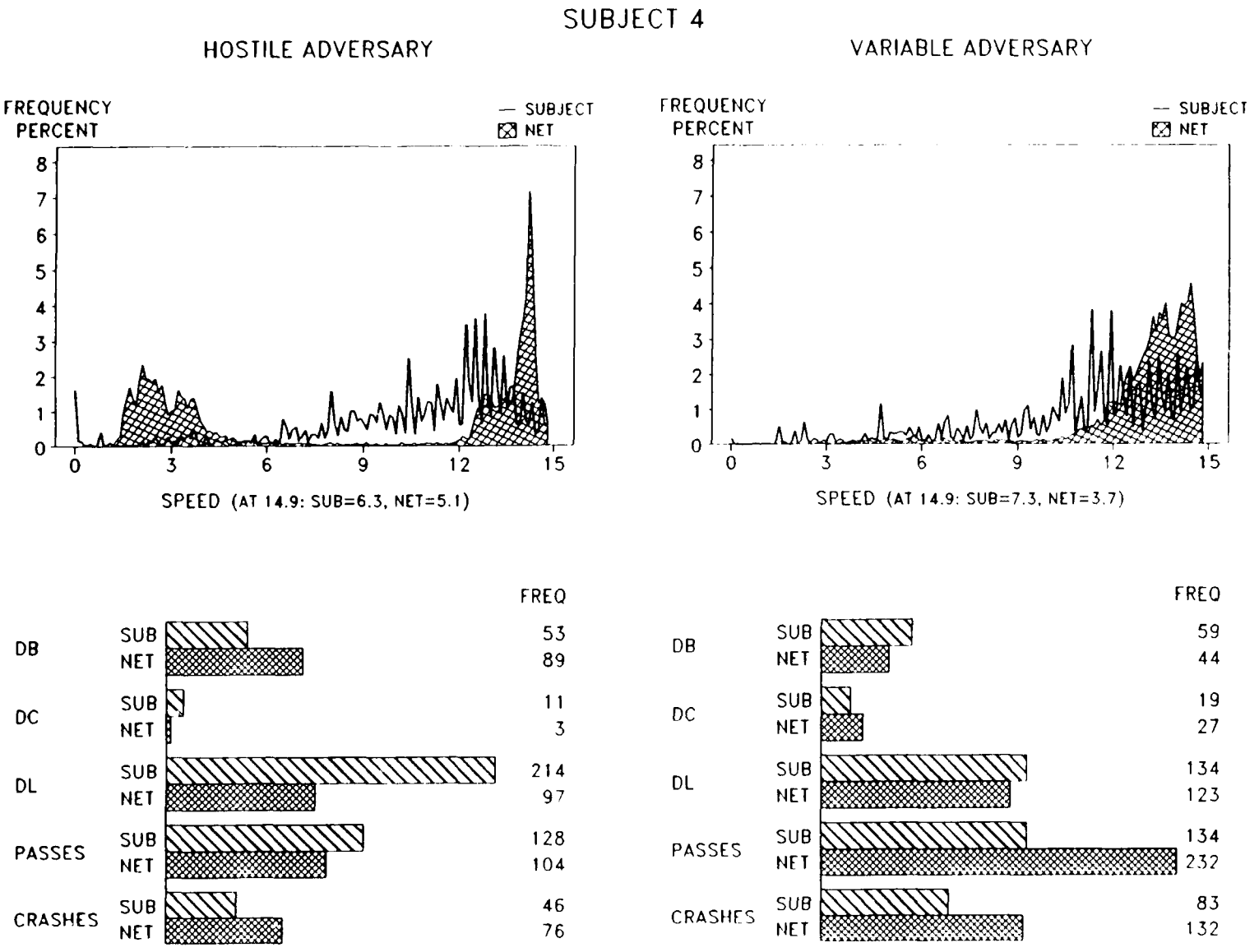


Figure A.7: Sub. 4 Speed Frequency and Data Points

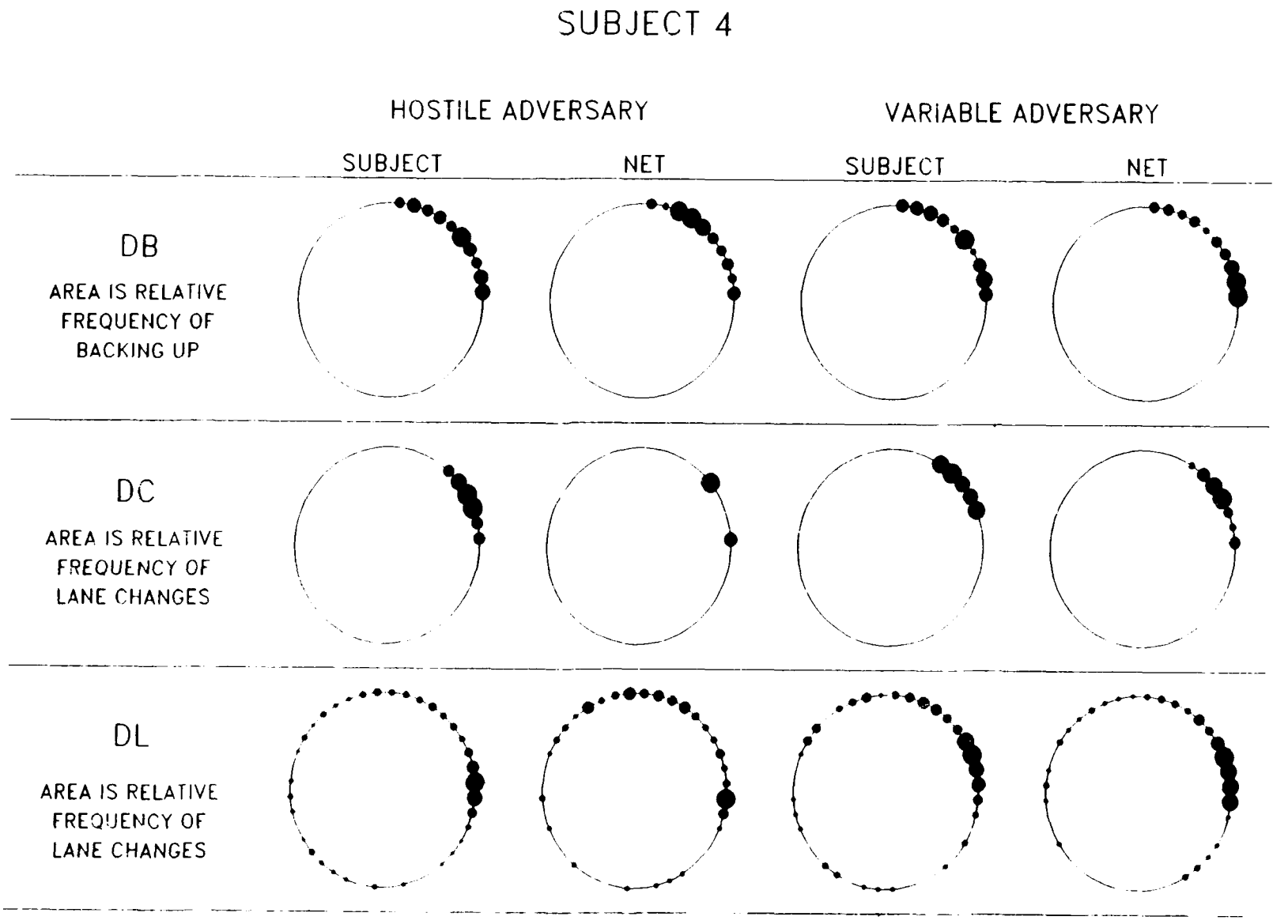


Figure A.8: Sub. 4 Relative Frequency of Data

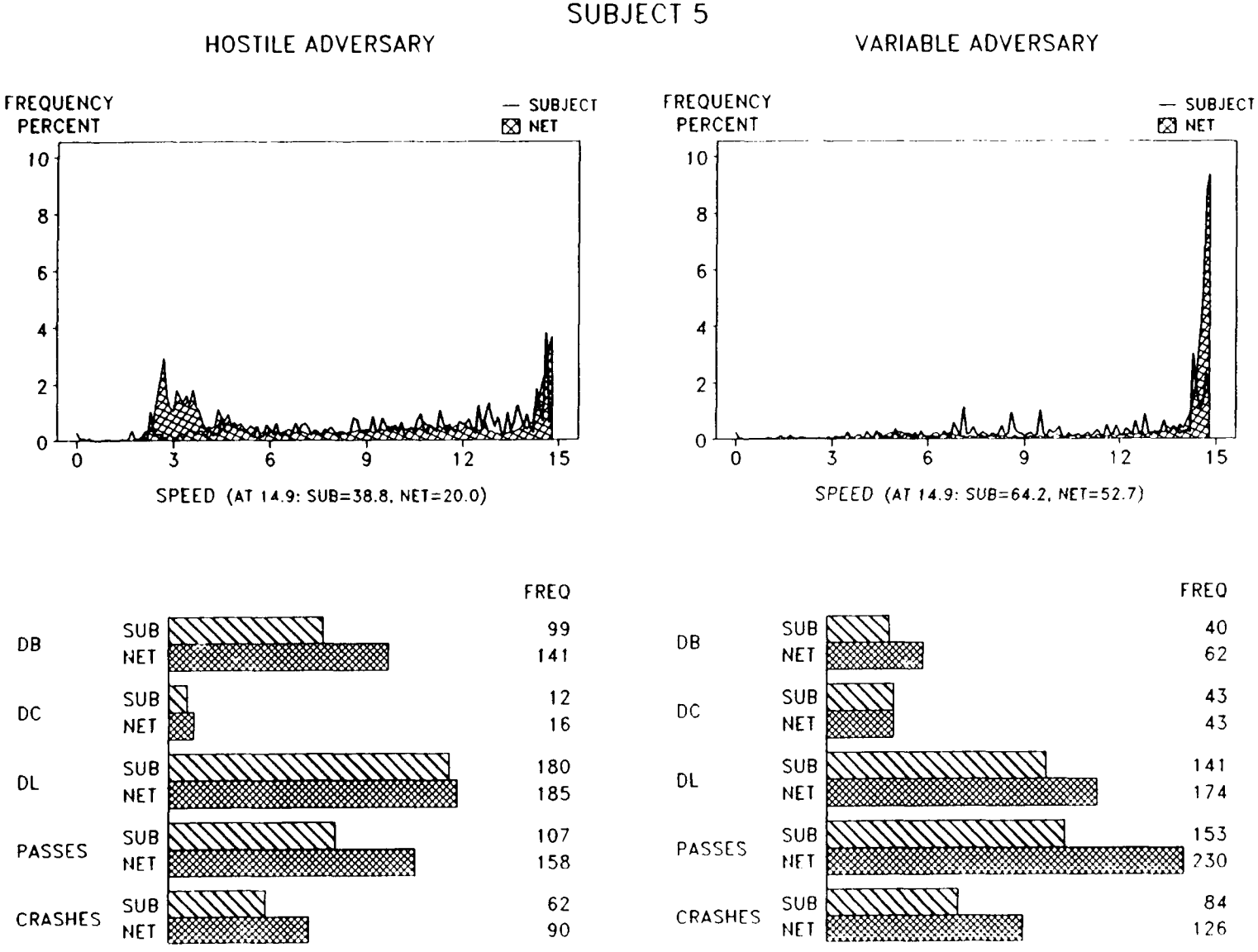


Figure A.9: Sub. 5 Speed Frequency and Data Points

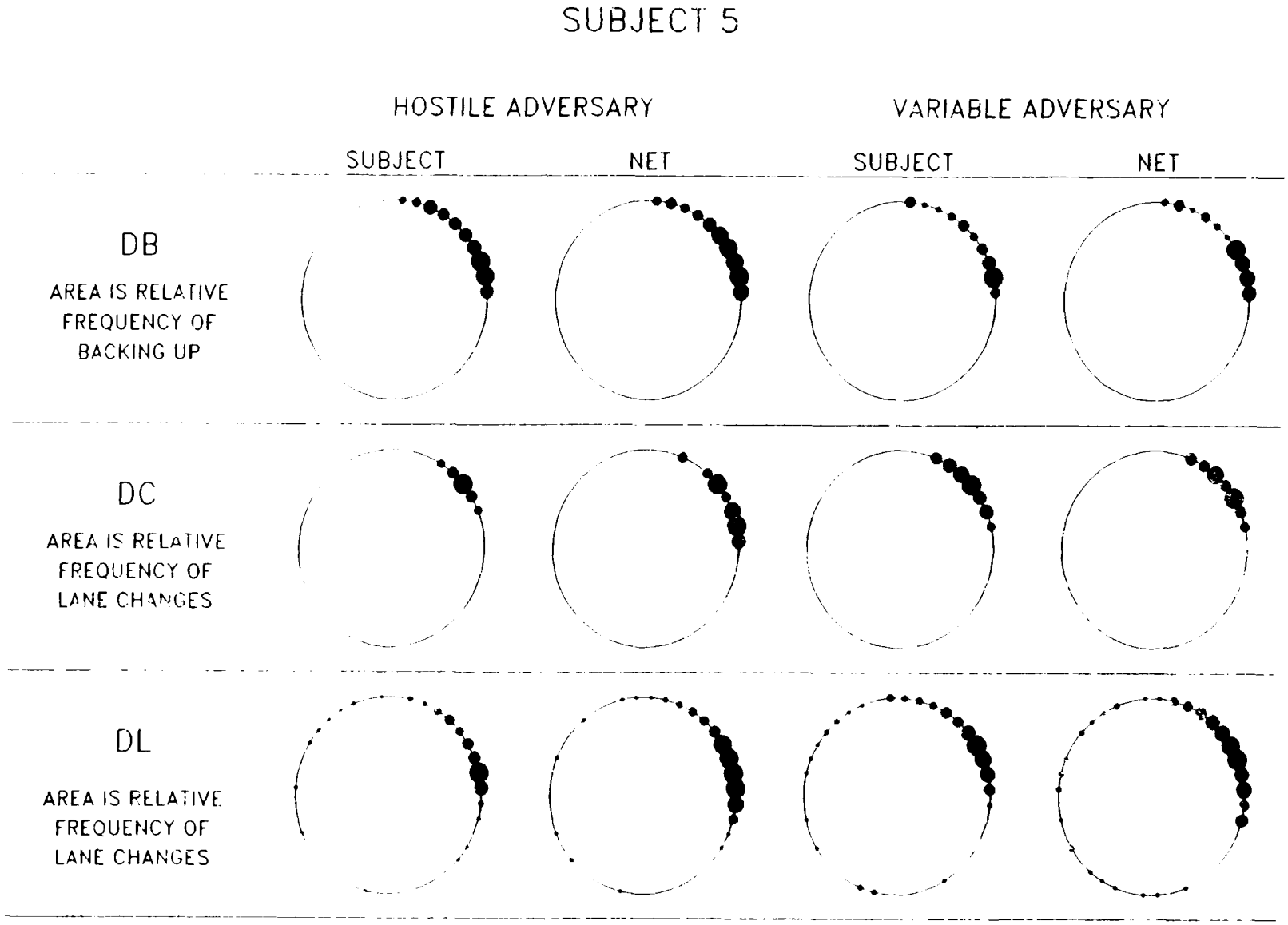


Figure A.10: Sub. 5 Relative Frequency of Data

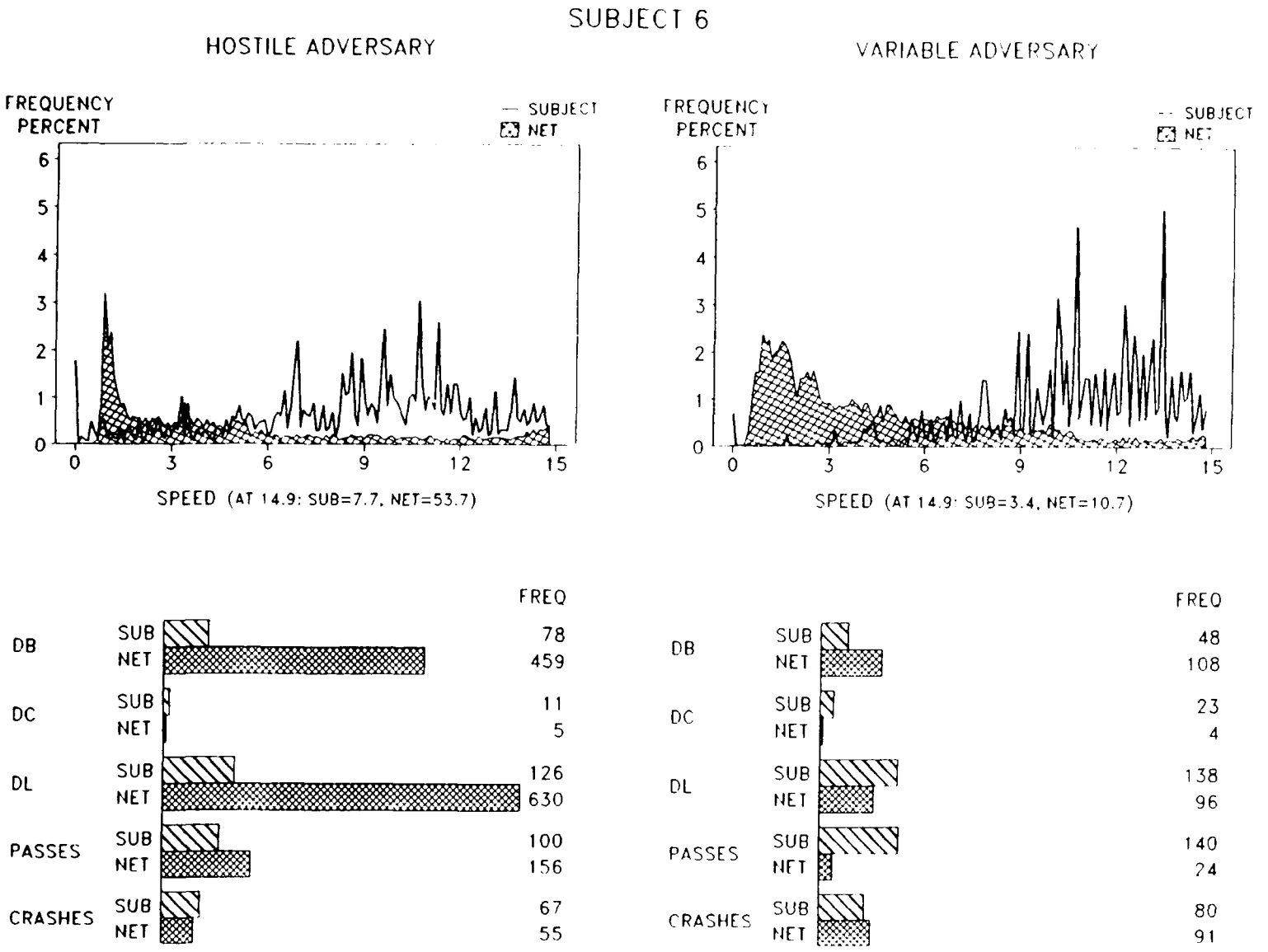


Figure A.11: Sub. 6 Speed Frequency and Data Points

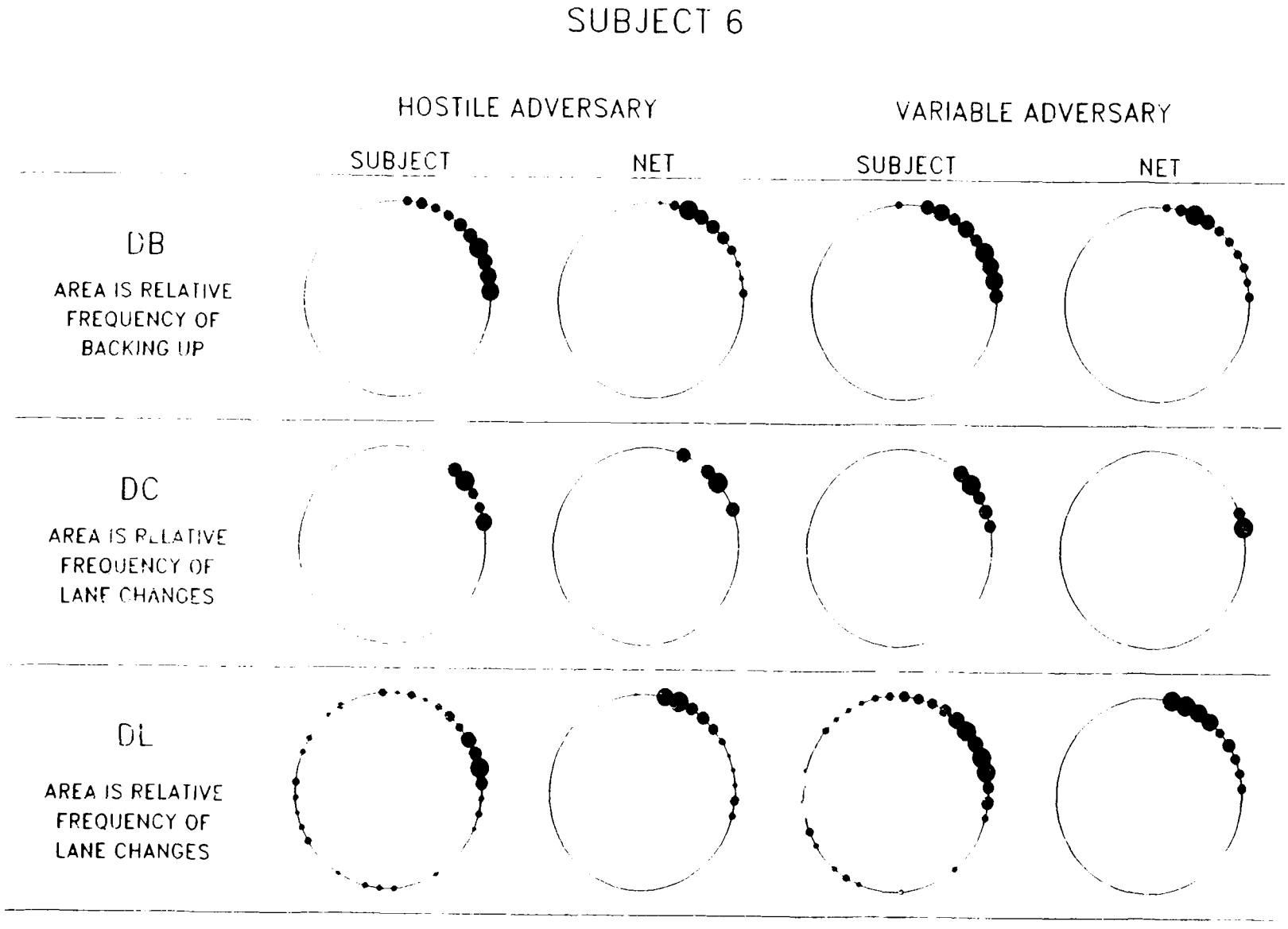


Figure A.12: Sub. 6 Relative Frequency of Data

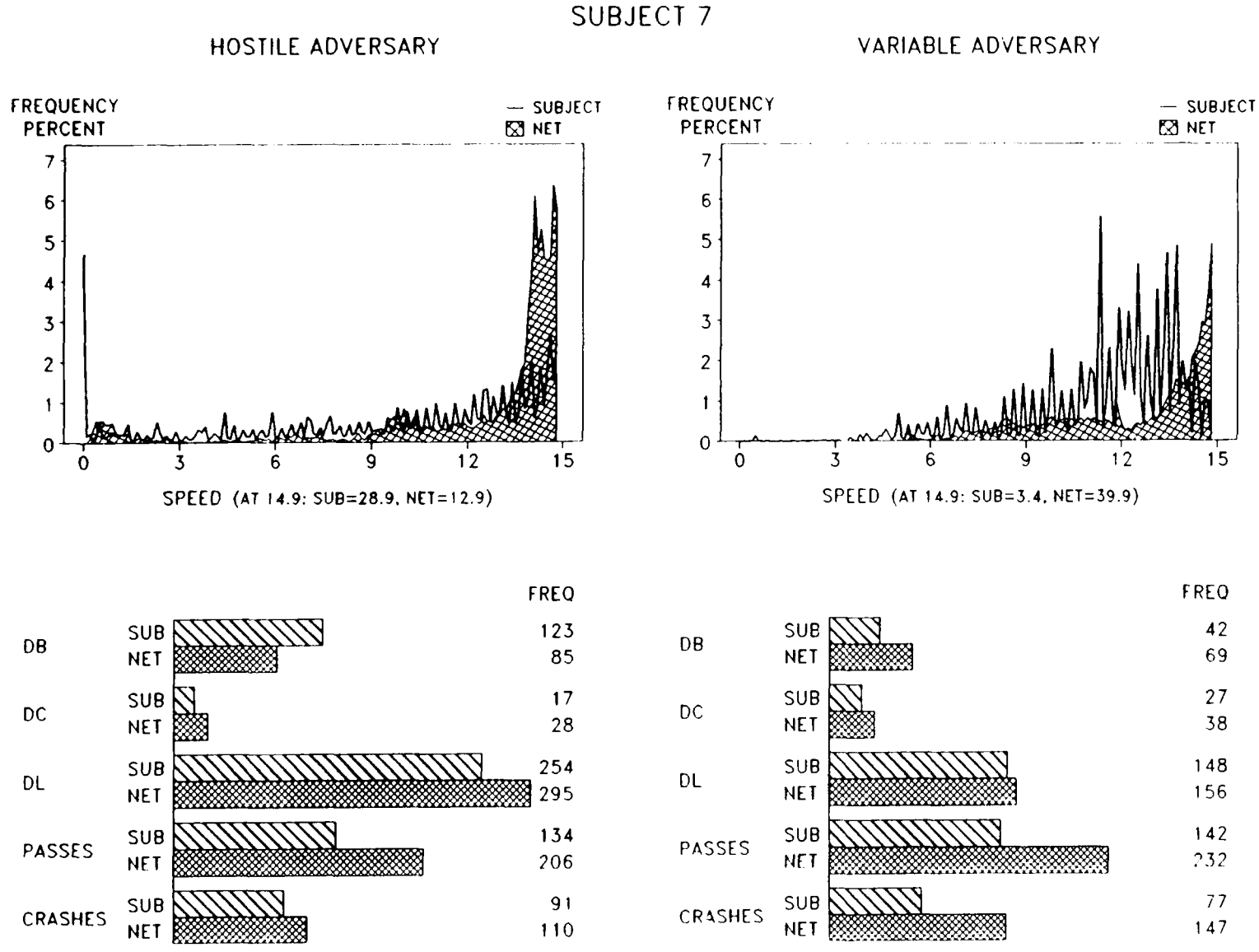


Figure A.13: Sub. 7 Speed Frequency and Data Points

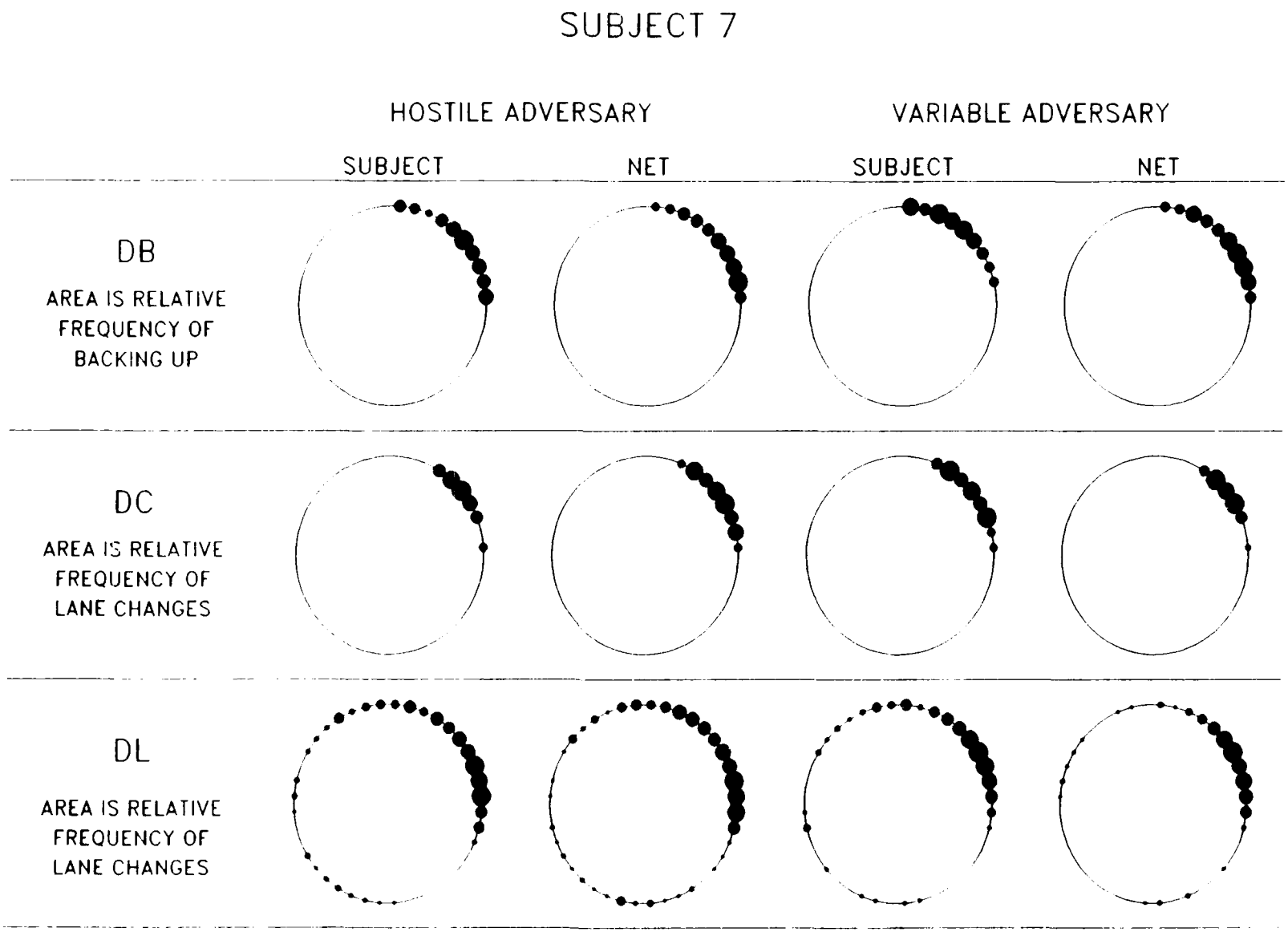


Figure A.14: Sub. 7 Relative Frequency of Data

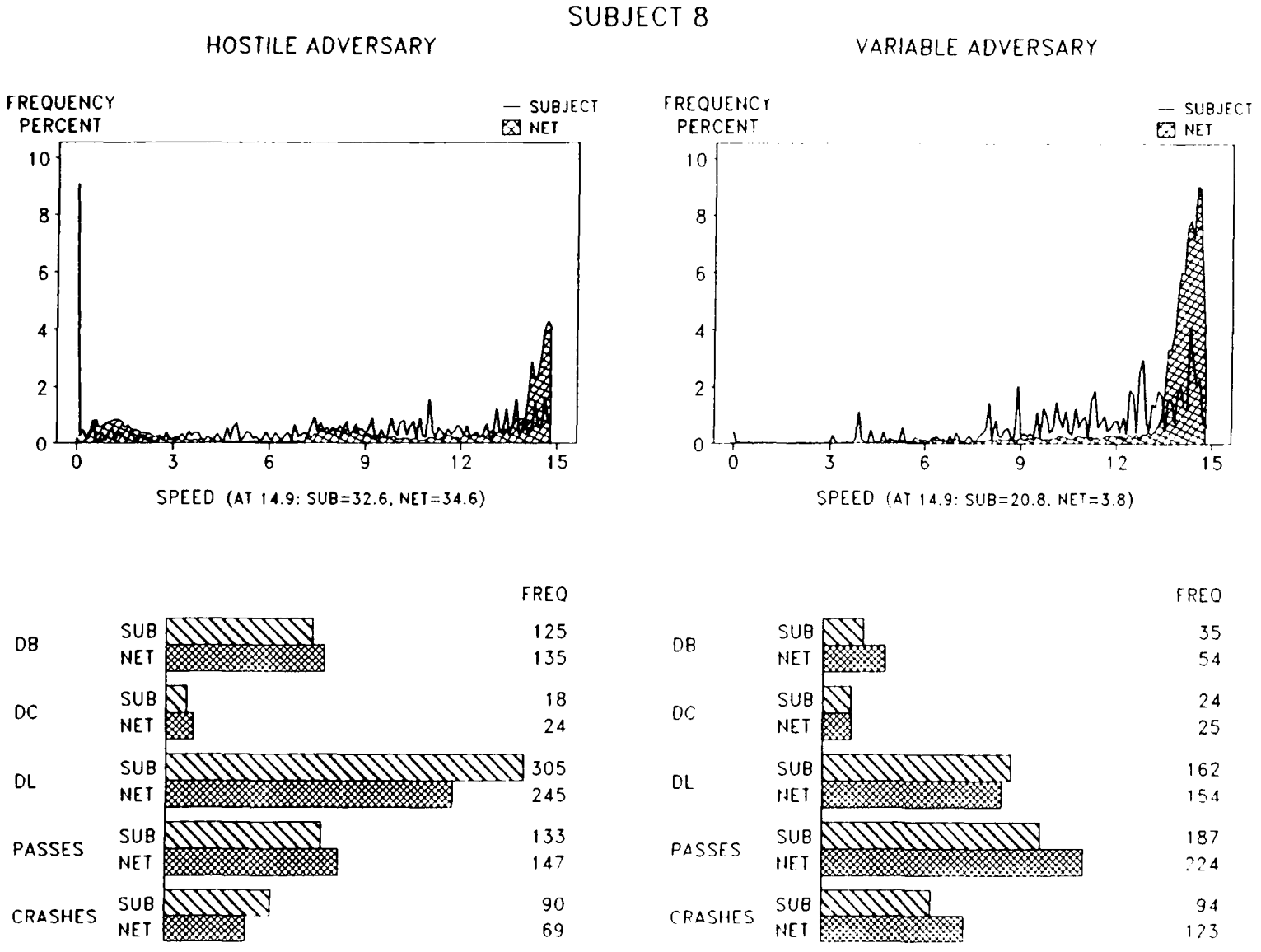


Figure A.15: Sub. 8 Speed Frequency and Data Points

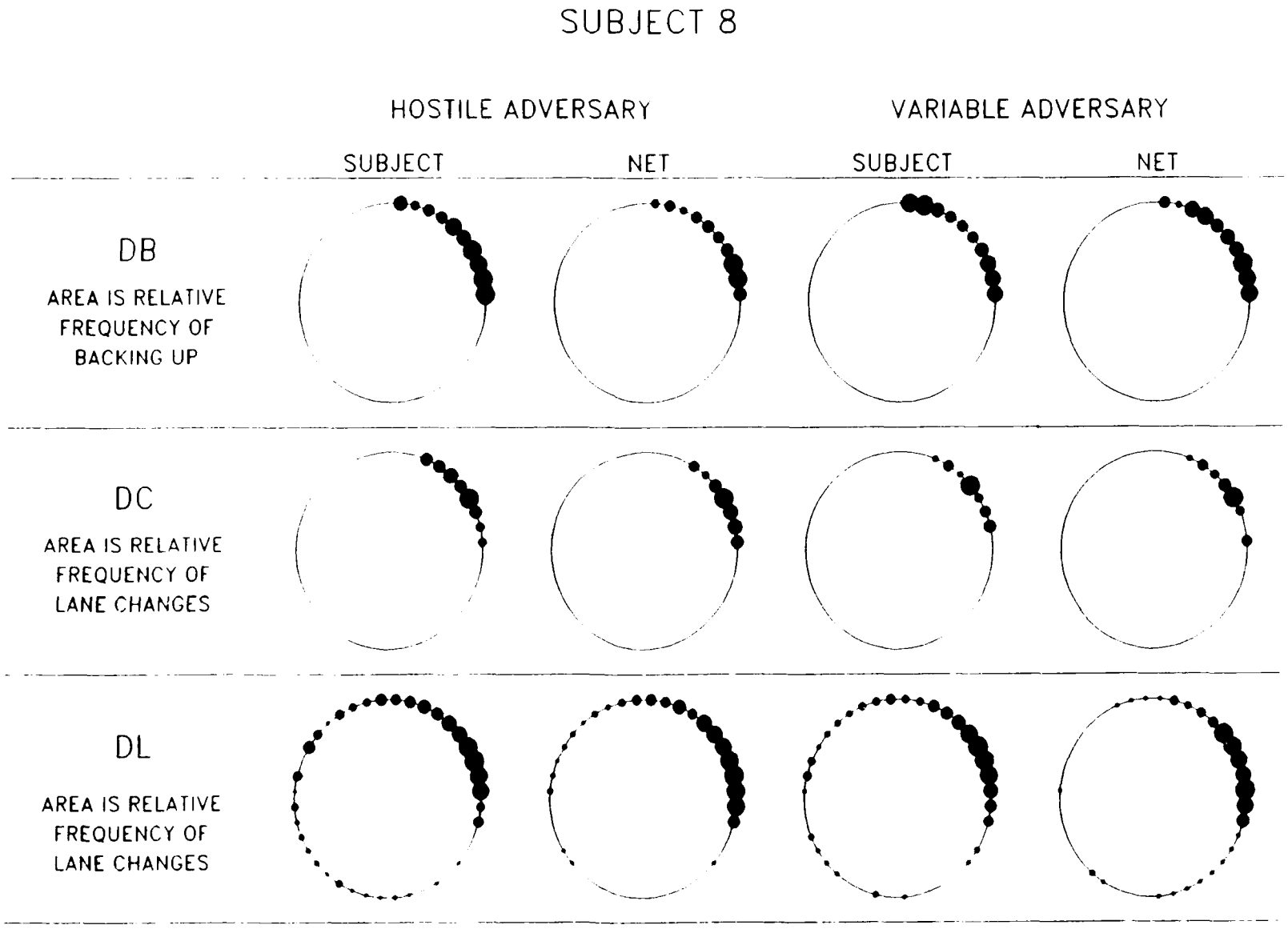


Figure A.16: Sub. 8 Relative Frequency of Data

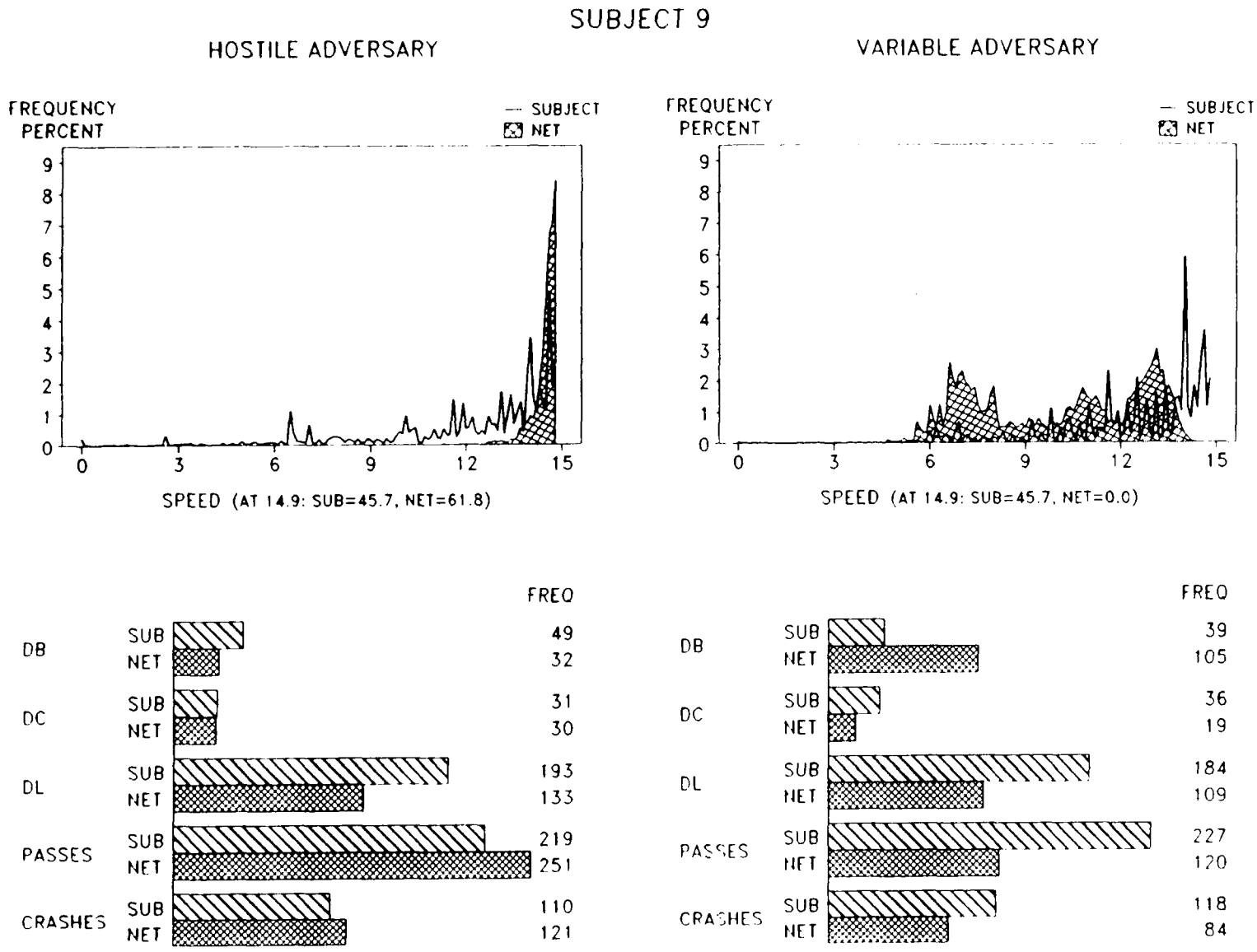


Figure A.17: Sub. 9 Speed Frequency and Data Points

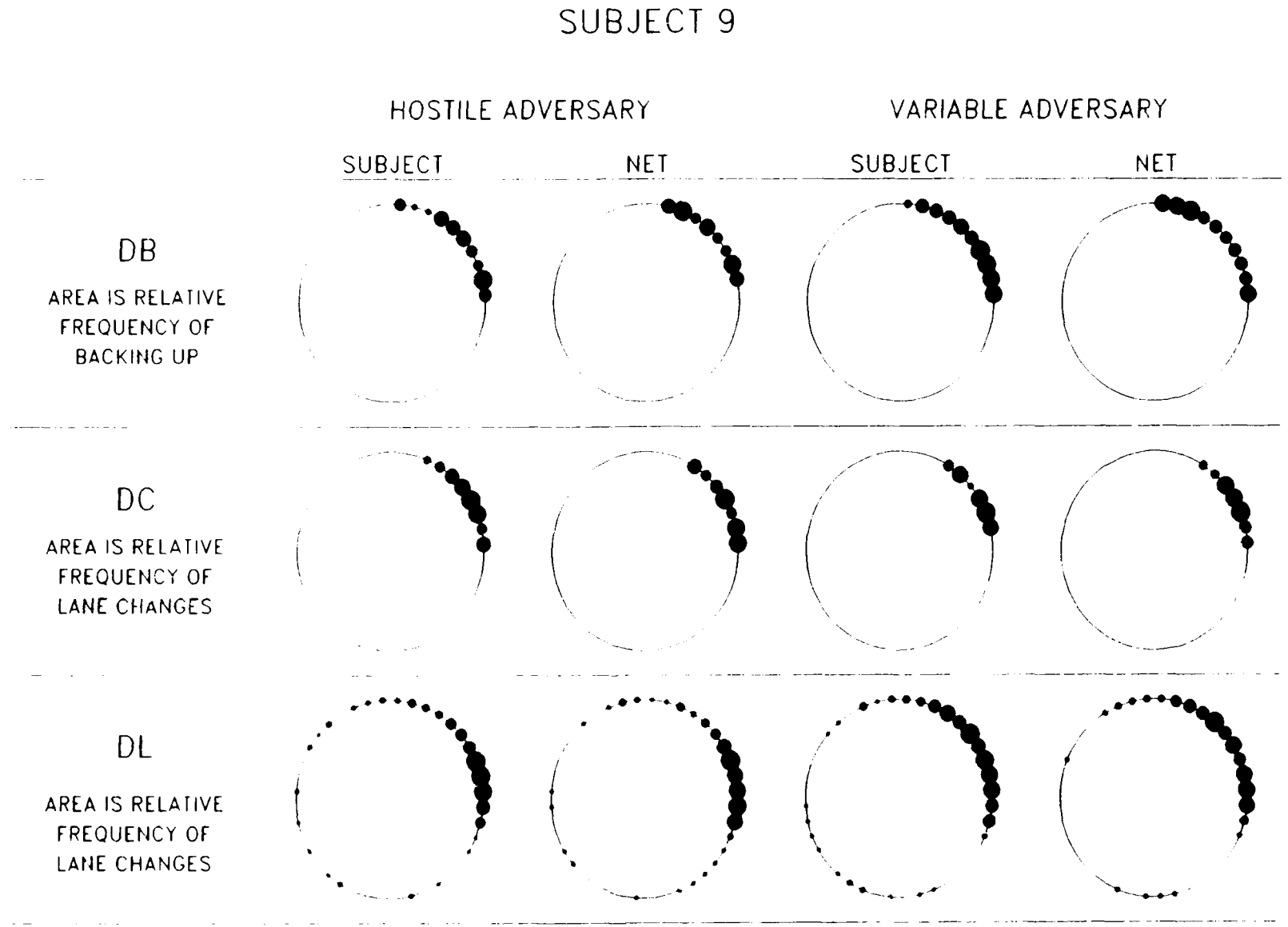


Figure A.18: Sub. 9 Relative Frequency of Data

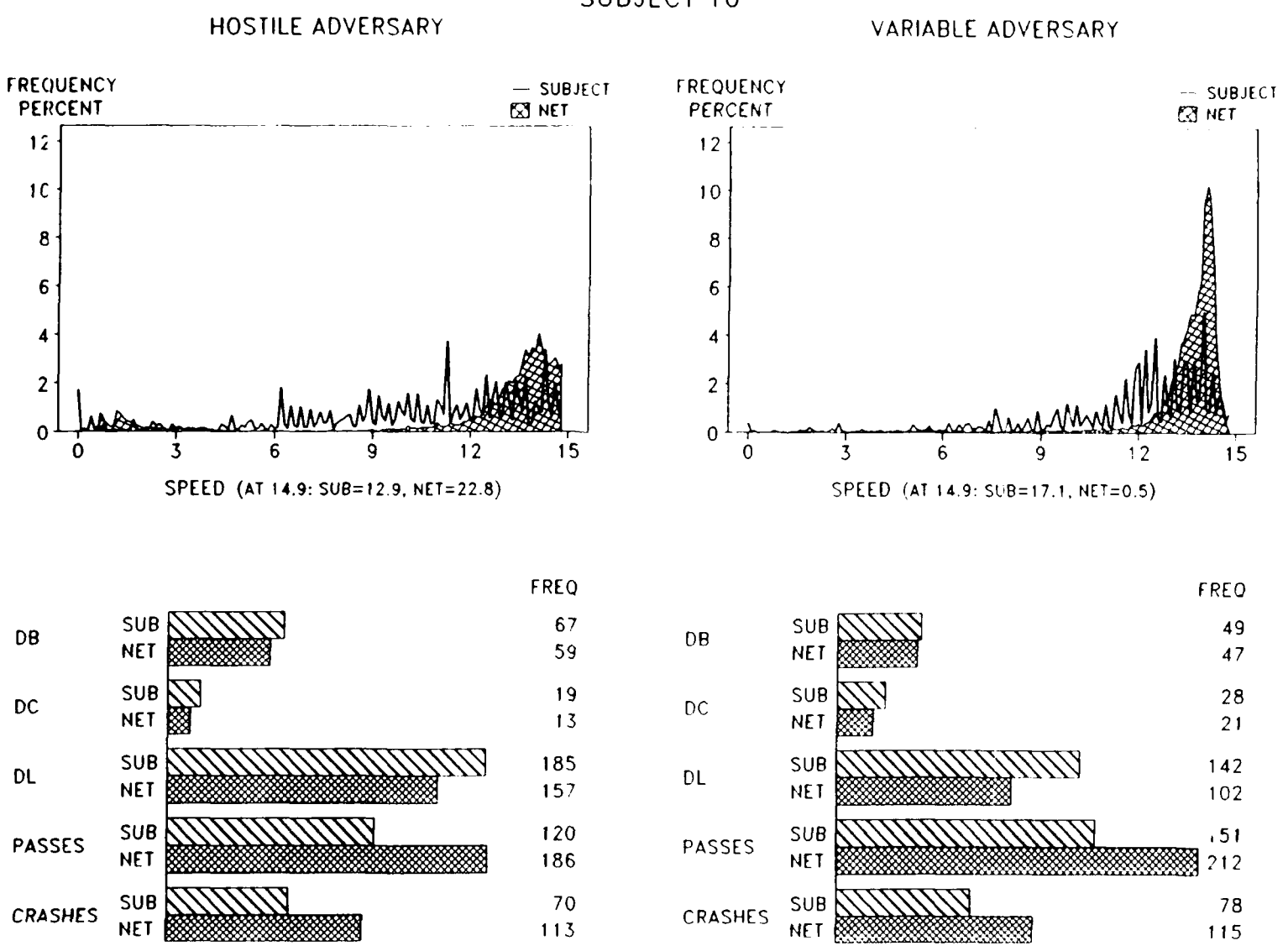


Figure A.19: Sub. 10 Speed Frequency and Data Points

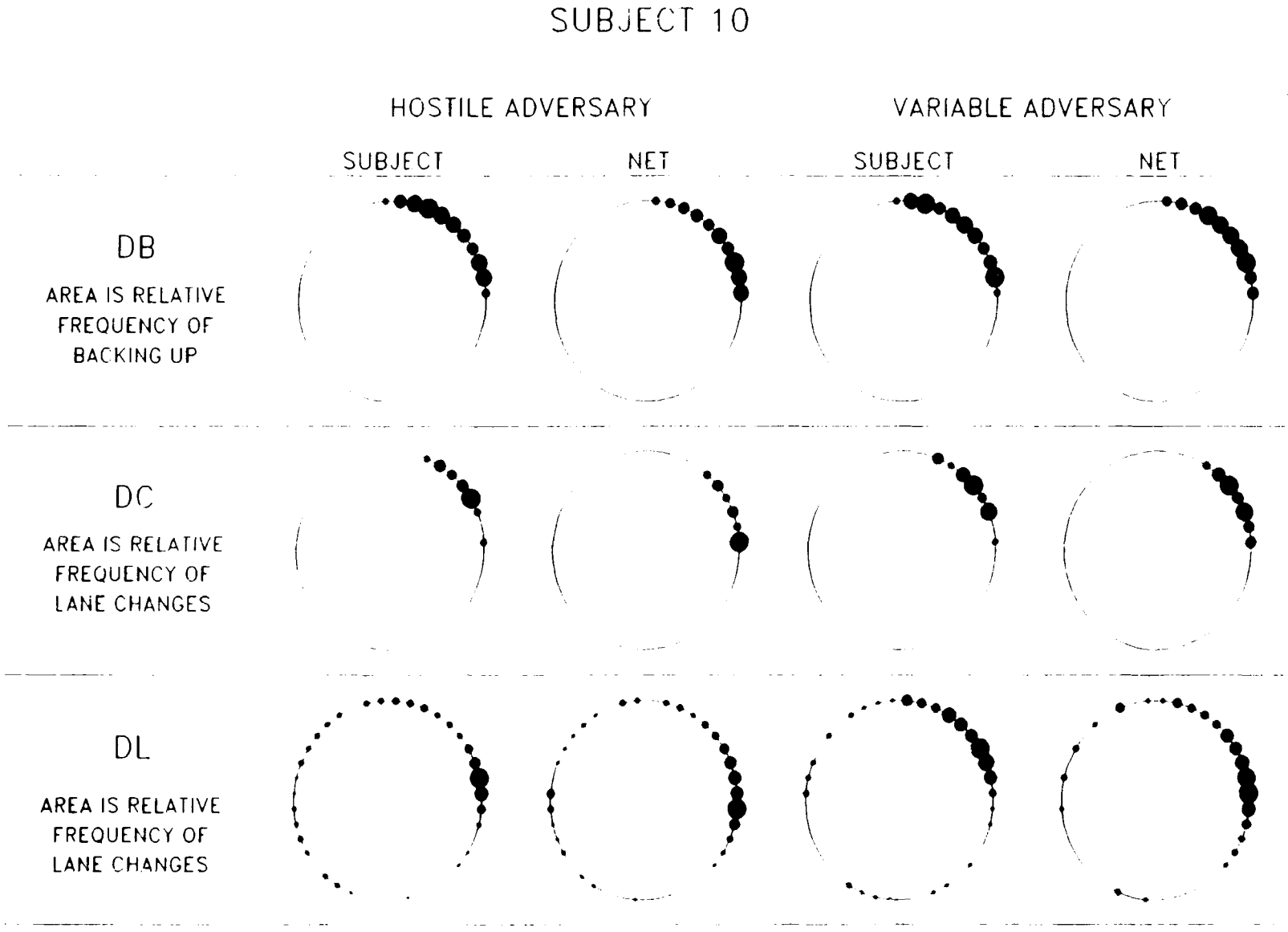


Figure A.20: Sub. 10 Relative Frequency of Data

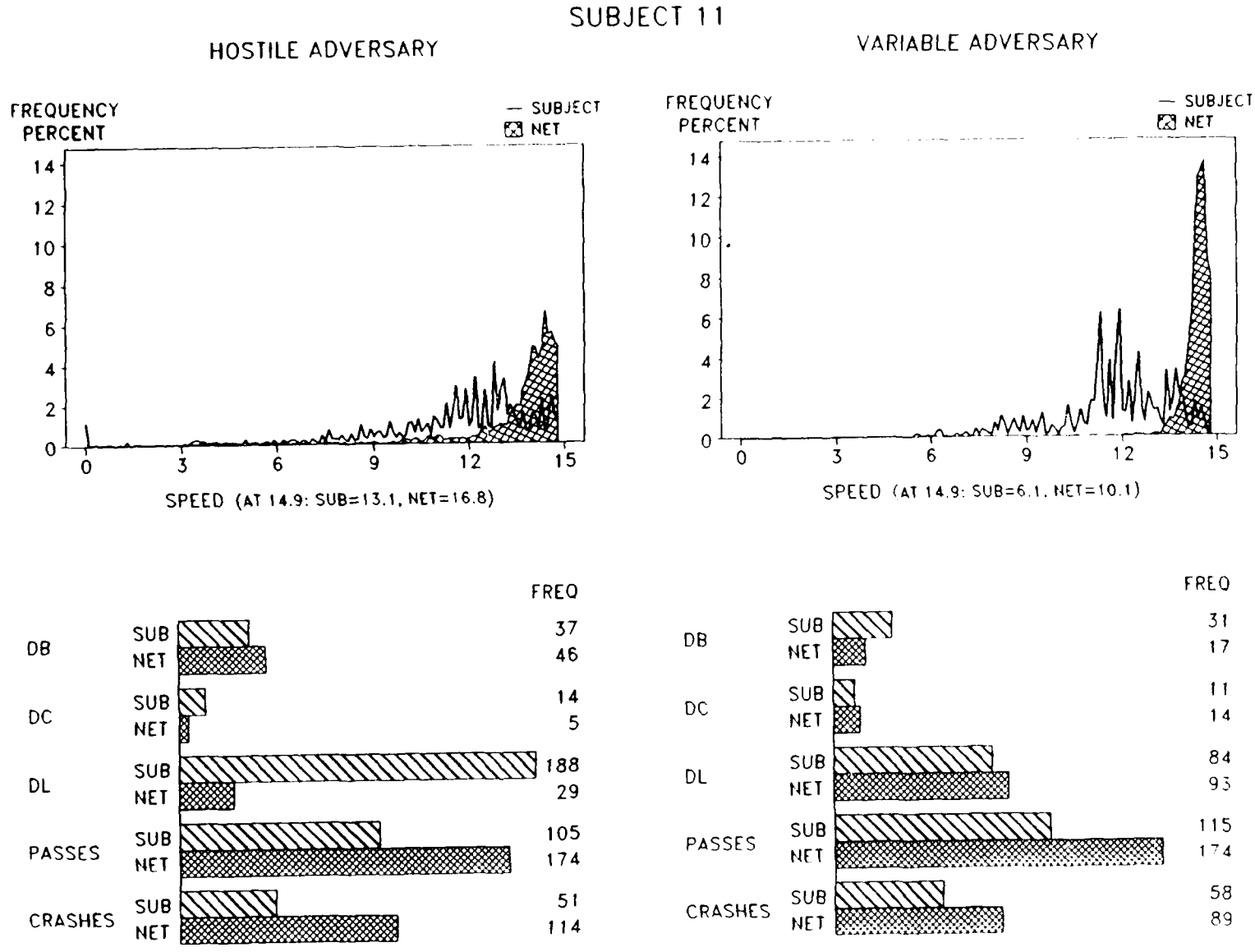


Figure A.21: Sub. 11 Speed Frequency and Data Points

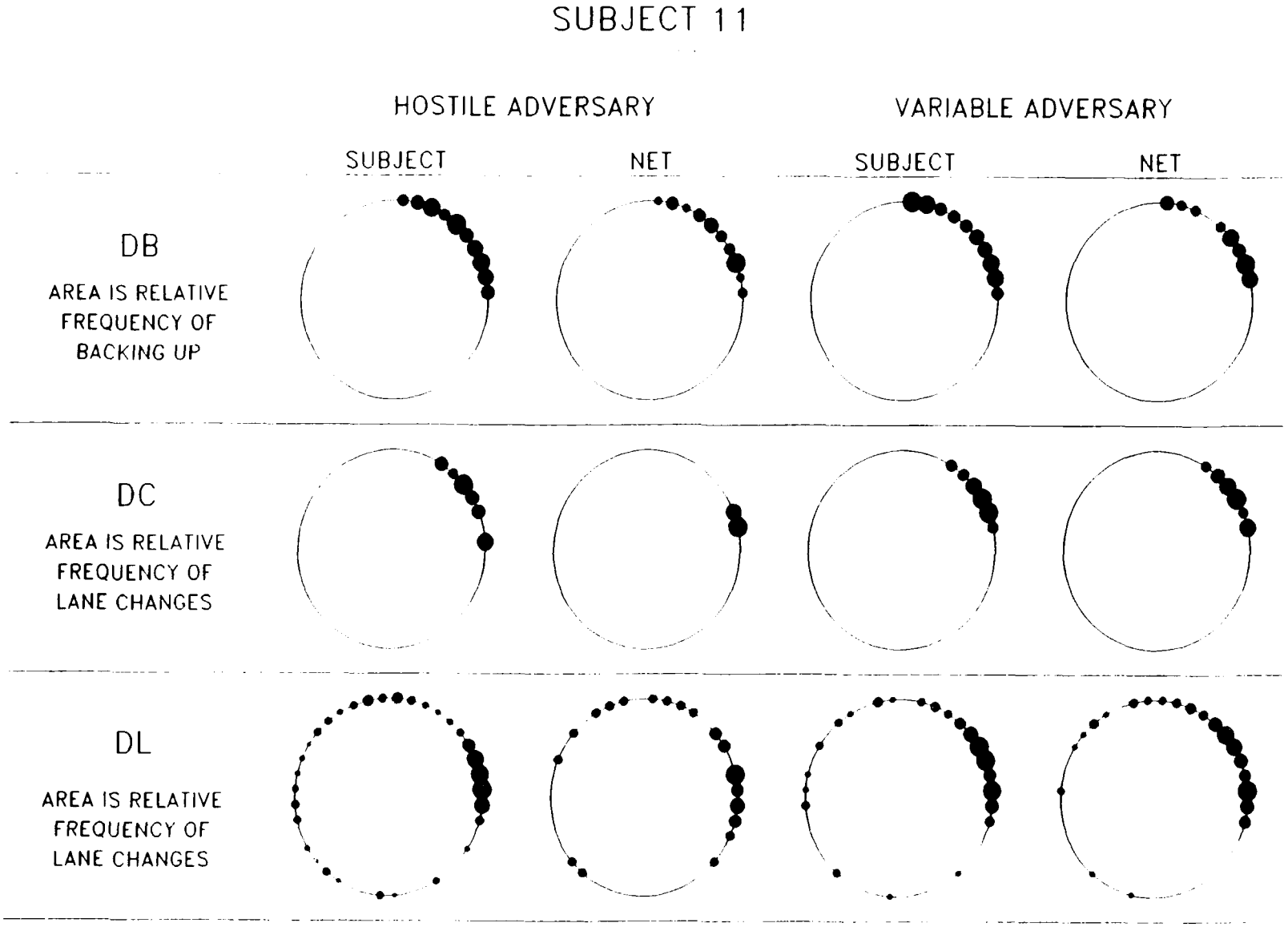


Figure A.22: Sub. 11 Relative Frequency of Data

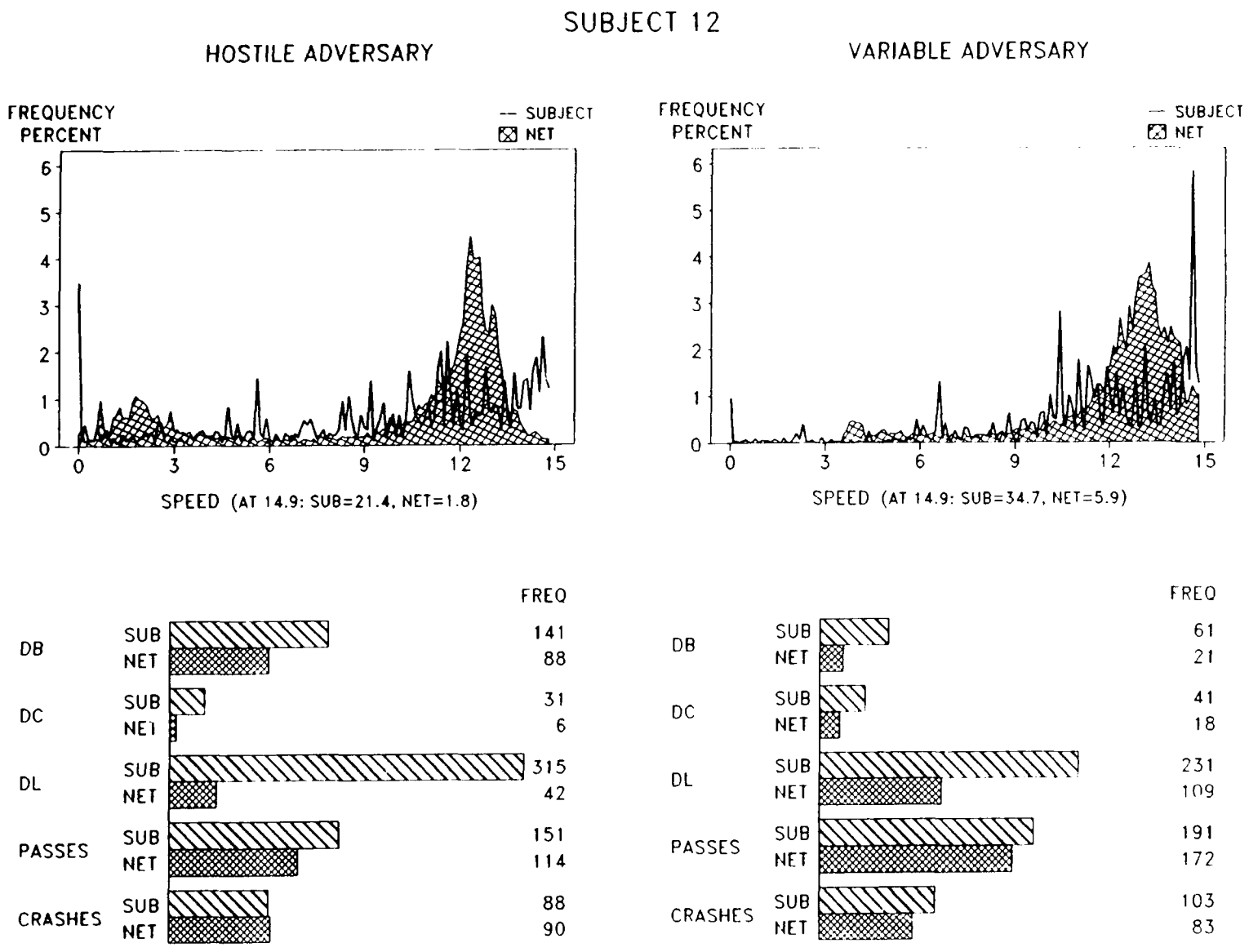


Figure A.23: Sub. 12 Speed Frequency and Data Points

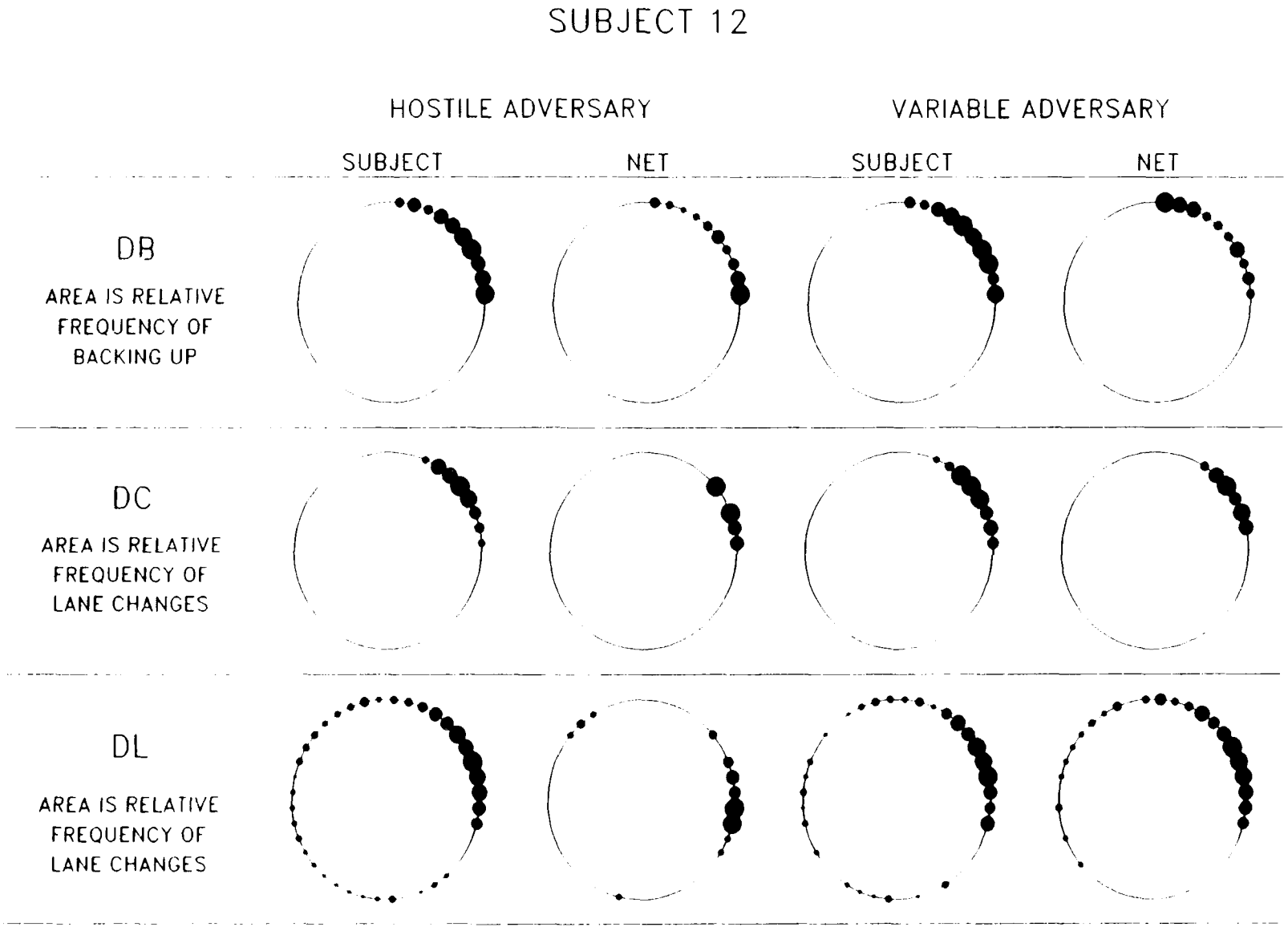


Figure A.24: Sub. 12 Relative Frequency of Data

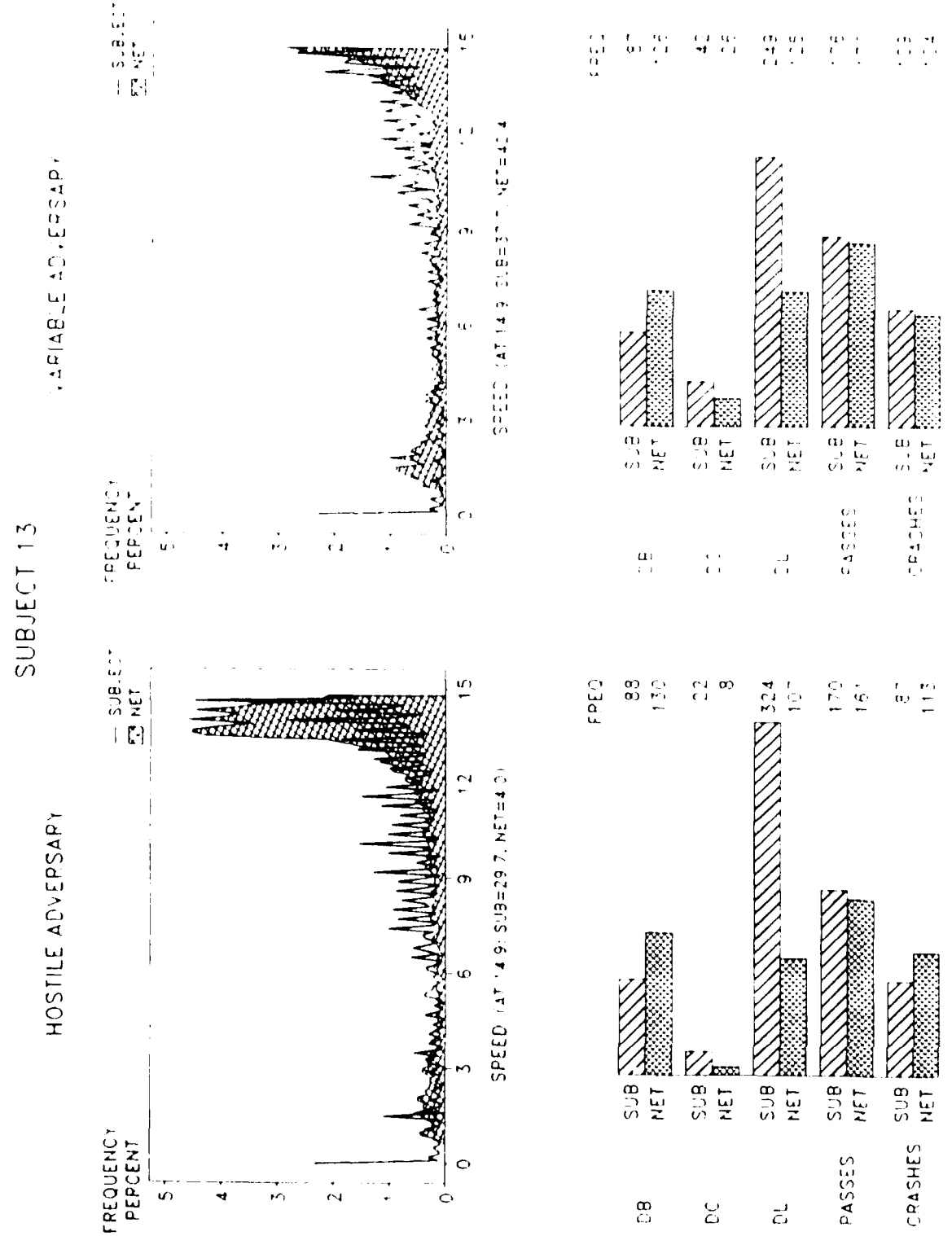


Figure A.25: Sub. 13 Speed Frequency and Data Points

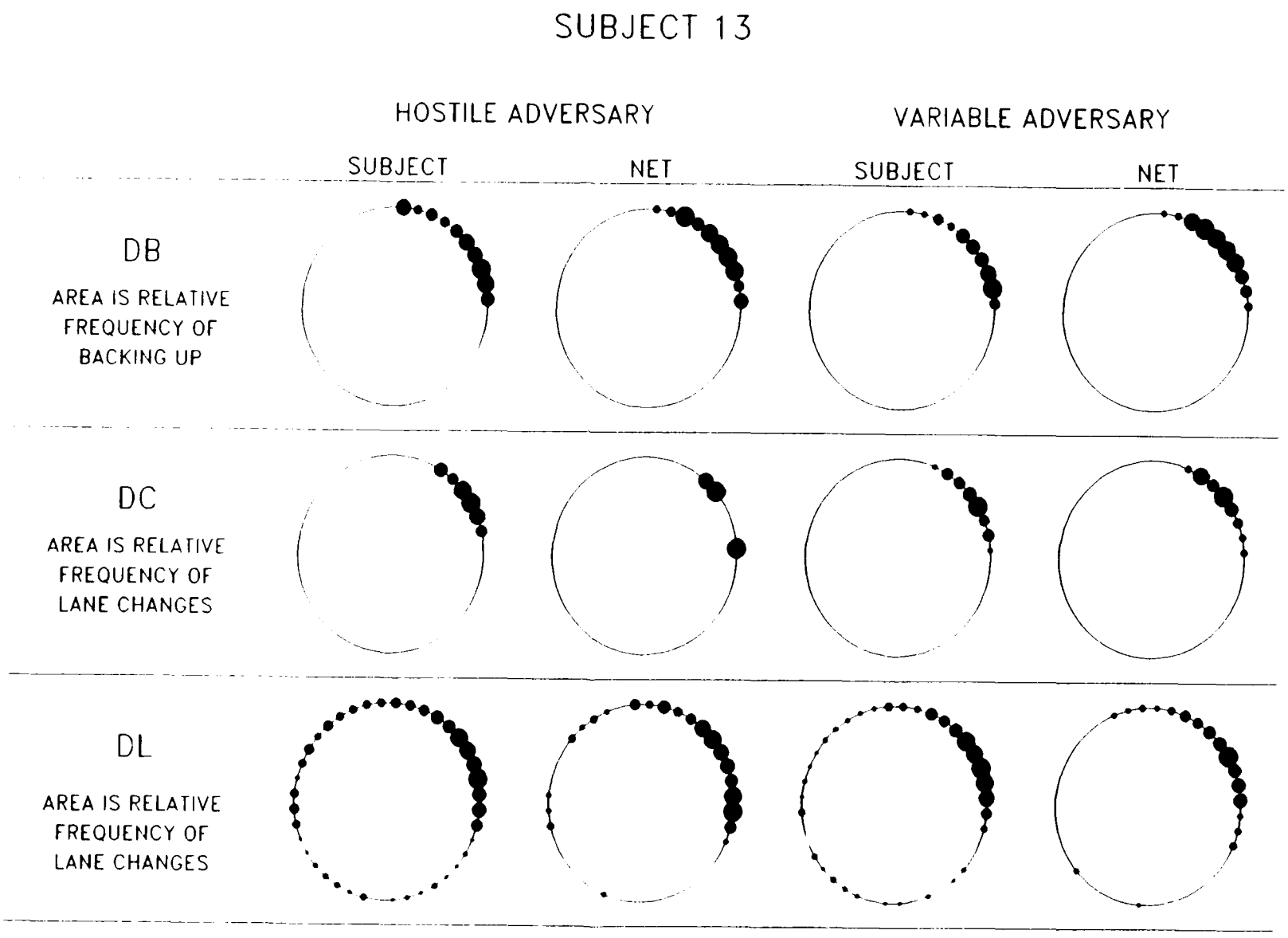


Figure A.26: Sub. 13 Relative Frequency of Data

Appendix B

Simulation C Source Code

```
/*****************************************************************************
        varicar.c

        Program to gather data in the "variable cars" case.

        Requires:
                init.c
                graph.c
                drive.c
                randvary.c
                util.c

*****************************************************************************/
#include "cars.h"
#include "carvars.h"

main()
{

        int i, j, k, lswitch, oldscore, score, testnr, testype, flag[2];
        static int bump[MYCAR] = {0, 0, 0, 0};
        float temp;
        FILE *out, *testin;
        long endtime, timer;
        char name[50], fname[50];
        char *path, *trainext, *testext, *title;
        extern FILE *fopen();
        extern float normal();

        path = "data/";
        trainext = ".sev";
        testext = ".ssv";
        title = "VARIABLE SPEED AND LANE";
        initvar();
        printf("Enter subject's name ... ");
        scanf("%s", name);
        i = copy(path, fname, 0, 6);
        i = copy(name, fname, i, 50);
        i = copy(trainext, fname, i, 5);
        out = fopen(fname, "w");
        if(out == 0) exit(0);
        initscreen();
        header(title, length(title), 0);
        for(i=0; i<NUMCARS; ++i)
                putcar(i, 0, &score);
        putmarker();
        score = 0;
        oldscore = 0;
        while((getbutton(LEFTMOUSE)) != 1)   /* Begin familiarization session */
        {
                for(j=0; j<DELAY; ++j){}
                for(j=0; j<DELAY; ++j){}
                for(i=0; i<MYCAR; ++i)
                {
```

```
                varyspeed(i);
                lswitch = varylane(i);
                putcar(i, lswitch, &score);
        }
        putmarker();
        idrive();
        for(i=0; i<MYCAR; ++i)
        {
                if(collision(i) && !bump[i])
                {
                        score -= 3;
                        bump[i] = 1;
                }
                if(!collision(i)) bump[i] = 0;
        }
        if(oldscore != score)
        {
                scoreboard((CENTERX + 290), (CENTERX + 299),
                        (CENTERY - 100 + (2 * score)),
                        (CENTERY - 100 + (2 * oldscore)),
                        (CENTERY - 100));
                oldscore = score;
        }
}
footer();          /*  Set up for Data Run  */
score = 0;
scoreboard((CENTERX + 290), (CENTERX + 299),
        (CENTERY - 100 + (2 * score)),
        (CENTERY - 100 + (2 * oldscore)),
        (CENTERY - 100));
oldscore = 0;
time(&timer);
endtime = timer + END;
while(time(&timer) < endtime)   /*Begin Continuous Data collection  */
{
        solidbox((CENTERX + 401), (CENTERX + 410), (CENTERY + 100),
             (CENTERY + 100 - (int)(((float)(END - endtime + timer)
               / (float)END) * 200)), BLACK);
        for(j=0; j<DELAY; ++j){}
        for(i=0; i<MYCAR; ++i)
        {
                varyspeed(i);
                lswitch = varylane(i);
                putcar(i, lswitch, &score);
        }
        putmarker();
        idrive();
        for(i=0; i<MYCAR; ++i)
        {
                fprintf(out, "%f ", v[i]);
                fprintf(out, "%f %d ", angle[i], lane[i]);
                if(collision(i) && !bump[i])
                {
                        score -= 3;
```

```
                        bump[i] = 1;
                }
                if(!collision(i)) bump[i] = 0;
        }
        fprintf(out, "%f %d\n", v[MYCAR], lane[MYCAR]);
        if(oldscore != score)
        {
                scoreboard((CENTERX + 290), (CENTERX + 299),
                        (CENTERY - 100 + (2 * score)),
                        (CENTERY - 100 + (2 * oldscore)),
                        (CENTERY - 100));
                oldscore = score;
        }
}
fprintf(out, "%f %f %d", 0.0, 0.0, 0);
fclose(out);          /*  End Continuous data collection  */

i = copy(path, fname, 0, 6);     /*  Set up for setup data collection  */
i = copy(name, fname, i, 50);
i = copy(testext, fname, i, 5);
out = fopen(fname, "w");
if(out == 0) exit(-1);

testin = fopen("testcar.dat", "r");
if(testin == 0) exit(-1);

lane[0] = 0;
lane[1] = 1;
fscanf(testin, "%d", &testnr);
for(k=0; k<testnr; ++k)     /*  Begin setup data collection  */
{
        initscreen();
        header(title, length(title), 1);
        unscore();
        solidbox((CENTERX + 401),  (CENTERX + 410), (CENTERY + 100),
             (CENTERY + 100 - (int)(((float)(k - 1)
                 / (float)testnr) * 200)), BLACK);
        fscanf(testin, "%d", &testype);
        for(i=0; i<2; ++i)
                fscanf(testin, "%f", &v[i]);
        for(i=0; i<2; ++i)
        {
                fscanf(testin, "%f", &angle[i]);
                if(testype < 8) angle[i] = normal(angle[i], .05);
        }
        for(i=0; i<2; ++i)
                putcar(i, 0, &score);
        lane[MYCAR] = 0;
        angle[MYCAR] = 0.0;
        putmycar(BLUE);
        putmarker();
        while((getbutton(LEFTMOUSE)) != 1){}
        flag[1] = 0;
        flag[0] = 0;
```

```
                fprintf(out, "%d\n", testype);
                while(flag[1] == 0 || flag[0] == 0)
                {
                        for(j=0; j<DELAY; ++j){}
                        for(i=0; i<2; ++i)
                        {
                                score = 0;
                                putcar(i, 0, &score);
                                if(score == 1) flag[i] = 1;
                        }
                        putmarker();
                        idrive();
                        for(i=0; i<2; ++i)
                                fprintf(out, "%f %f %d ",
                                        v[i], angle[i], lane[i]);
                        fprintf(out, "%f %d\n", v[MYCAR], lane[MYCAR]);
                }
        fprintf(out, "%f %f %d\n", 0.0, 0.0, 0);
        }
        fclose(out);

        color(BLACK);
        ginit();
        clear();
}
```

```
/*****************************************************************************
        hostilcar.c

        Program to train and gather data in the "hostile cars" case.

        Requires:
                init.c
                graph.c
                drive.c
                randvary.c
                hostile.c
                util.c

*****************************************************************************/
#include "cars.h"
#include "carvars.h"

main()
{

        int i, j, k, lswitch, oldscore, score, testype, testnr, flag[2];
        static int bump[MYCAR] = {0, 0, 0, 0};
        static int hostility[MYCAR] = {0, 0, 0, 0};
        FILE *out, *testin;
        long endtime, timer;
        char name[50], fname[50];
        char *path, *trainext, *testext, *title;
        extern FILE *fopen();
        extern float normal();

        path = "data/";
        trainext = ".seh";
        testext = ".ssh";
        title = "THE HOSTILE TAKEOVER";
        initvar();
        printf("Enter subject's name ... ");
        scanf("%s", name);
        i = copy(path, fname, 0, 6);
        i = copy(name, fname, i, 50);
        i = copy(trainext, fname, i, 5);
        out = fopen(fname, "w");
        if(out == 0) exit(0);

        initscreen();
        header(title, length(title), 1);
        for(i=0; i<NUMCARS; ++i)
                putcar(i, 0, &score);
        putmarker();
        score = 0;
        oldscore = 0;
        while((getbutton(LEFTMOUSE)) != 1)  /* Begin familiarization run  */
        {
                for(j=0; j<DELAY; ++j){}  /* set screen animation speed  */
                for(j=0; j<DELAY; ++j){}
```

```
        for(i=0; i<MYCAR; ++i)
        {
                if((angle[i] < 0.1) && (lane[i] != lane[MYCAR]))
                {
                        lswitch = stoppass(i, &hostility[i]);
                        putcar(i, lswitch, &score);
                }
                else
                {
                        varyspeed(i);
                        lswitch = varylane(i);
                        hostility[i] = 0;
                        if((following[i] == -1) && (oldspeed[i] > 0.0))
                        {
                                v[i] = oldspeed[i];
                                oldspeed[i] = 0.0;
                        }
                        putcar(i, lswitch, &score);
                }
        }
        putmarker();
        idrive();
        for(i=0; i<MYCAR; ++i)
        {
                if(collision(i) && !bump[i])
                {
                        score -= 3;
                        bump[i] = 1;
                }
                if(!collision(i)) bump[i] = 0;
        }
        if(score != oldscore)
        {
                scoreboard((CENTERX + 290), (CENTERX + 299),
                        (CENTERY - 100 + (2 * score)),
                        (CENTERY - 100 + (2 * oldscore)),
                        (CENTERY - 100));
                oldscore = score;
        }
}
footer();   /* Set up for data collection  */
score = 0;
scoreboard((CENTERX + 290), (CENTERX + 299),
        (CENTERY - 100 + (2 * score)),
        (CENTERY - 100 + (2 * oldscore)),
        (CENTERY - 100));
oldscore = 0;
time(&timer);
endtime = timer + END;
while(time(&timer) < endtime)   /* Begin continuous data collection  */
{
        solidbox((CENTERX + 401), (CENTERX + 410), (CENTERY + 100),
             (CENTERY + 100 - (int)(((float)(END - endtime + timer)
               / (float)END) * 200)), BLACK);
```

```
        for(j=0; j<DELAY; ++j){}
        for(i=0; i<MYCAR; ++i)
        {
                if((angle[i] < 0.1) && (lane[i] != lane[MYCAR]))
                {
                        lswitch = stoppass(i, &hostility[i]);
                        putcar(i, lswitch, &score);
                }
                else
                {
                        varyspeed(i);
                        lswitch = varylane(i);
                        hostility[i] = 0;
                        if((following[i] == -1) && (oldspeed[i] > 0.0))
                        {
                                v[i] = oldspeed[i];
                                oldspeed[i] = 0.0;
                        }
                        putcar(i, lswitch, &score);
                }
        }
        putmarker();
        idrive();
        for(i=0; i<MYCAR; ++i)
        {
                fprintf(out, "%f ", v[i]);
                fprintf(out, "%f %d ", angle[i], lane[i]);
                if(collision(i) && !bump[i])
                {
                        score -= 3;
                        bump[i] = 1;
                }
                if(!collision(i)) bump[i] = 0;
        }
        fprintf(out, "%f %d\n", v[MYCAR], lane[MYCAR]);
        if(score != oldscore)
        {
                scoreboard((CENTERX + 290), (CENTERX + 299),
                        (CENTERY - 100 + (2 * score)),
                        (CENTERY - 100 + (2 * oldscore)),
                        (CENTERY - 100));
                oldscore = score;
        }
}
fprintf(out, "%f %f %d", 0.0, 0.0, 0);
fclose(out);  /*   End continuous data collection  */

i = copy(path, fname, 0, 6);
i = copy(name, fname, i, 50);
i = copy(testext, fname, i, 5);
out = fopen(fname, "w");
if(out == 0) exit(-1);

testin = fopen("testcar.dat", "r");
```

```
if(testin == 0) exit(-1);

fscanf(testin, "%d", &testnr);
for(k=0; k<testnr; ++k)   /*  Begin setup data collection  */
{
        initscreen();
        header(title, length(title), 1);
        unscore();
        solidbox((CENTERX + 401),  (CENTERX + 410), (CENTERY + 100),
             (CENTERY + 100 - (int)(((float)(k - 1)
                 / (float)testnr) * 200)), BLACK);
        fscanf(testin, "%d", &testype);
        for(i=0; i<2; ++i)
        {
                fscanf(testin, "%f", &v[i]);
                oldspeed[i] = 0.0;
                following[i] = -1;
        }
        for(i=0; i<2; ++i)
        {
                fscanf(testin, "%f", &angle[i]);
                if(testype < 8) angle[i] = normal(angle[i], 0.05);
        }
        lane[0] = 0;
        lane[1] = 1;
        for(i=0; i<2; ++i)
                putcar(i, 0, &score);
        lane[MYCAR] = 0;
        angle[MYCAR] = 0.0;
        putmycar(BLUE);
        putmarker();
        angle[2] = 0.75;   /*  not used  */
        angle[3] = 0.75;   /*  not used  */
        while((getbutton(LEFTMOUSE)) != 1){}
        flag[1] = 0;   /*  Passed Car 1 flag  */
        flag[0] = 0;   /*  Passed Car 2 flag  */
        fprintf(out, "%d\n", testype);
        while(flag[0] == 0 || flag[1] == 0)
        {
                for(j=0; j<DELAY; ++j){}
                for(i=0; i<2; ++i)
                {
                        if((angle[i] < 0.1) && (lane[i] != lane[MYCAR]))
                                lswitch = stoppass(i, &hostility[i]);
                        else
                        {
                                lswitch = varylane(i);
                                hostility[i] = 0;
                                if((following[i] == -1) &&
                                   (oldspeed[i] > 0.0))
                                {
                                v[i] = oldspeed[i];
                                oldspeed[i] = 0.0;
                                }
```

```
                        }
                        score = 0;
                        putcar(i, lswitch, &score);
                        if(score == 1) flag[i] = 1;
                }
                putmarker();
                idrive();
        for(i=0; i<2; ++i)
                fprintf(out, "%f %f %d ", v[i], angle[i], lane[i]);
        fprintf(out, "%f %d\n", v[MYCAR], lane[MYCAR]);
        }
    fprintf(out, "%f %f %d\n", 0.0, 0.0, 0);
    }
    fclose(out);
    color(BLACK);
    ginit();
    clear();
}
```

```
/*****************************************************************************
        randvary.c

        Contains functions to vary speed and lane of computer controlled
        cars.

*****************************************************************************/
#include "cars.h"
#include "carextvar.h"

varyspeed(car)
int car;
{
        extern float normal();

        if(drand48() > .98)
                v[car] = normal(v[car], 2.0);
        if(v[car] > 12.0)v[car] = 12.0;
        if(v[car] < 3.0)v[car] = 3.0;
}

int varylane(car)
int car;
{
        int i;
        int lswitch = -1;
        int slowdown = 0;
        float fwdangle, aftangle;

if((following[car] > -1) && (lane[following[car]] != lane[car]))
{
        following[car] = -1;
        v[car] = oldspeed[car];
        oldspeed[car] = 0.0;
}
if(following[car] > -1)
{
        v[car] = v[following[car]];
        for(i=0; i<NUMCARS; ++i)
        {
                if((i != car) && (i != following[car]))
                {
                        refangles(&fwdangle, &aftangle, car, i);
                        if(safetopass(fwdangle, aftangle, angle[car],
                                        lane[car], lane[i]) == 0)
                                slowdown = 1;
                }
        }
        if(slowdown == 0)
        {
                v[car] = oldspeed[car];
                oldspeed[car] = 0.0;
                lswitch = 1;
                following[car] = -1;
```

```
        }
}
else
{
        for(i=0; i<NUMCARS; ++i)
        {
                if(i != car)
                {
                        refangles(&fwdangle, &aftangle, car, i);
                        if(needtopass(fwdangle, angle[car], lane[car], lane[i],
                                        v[car], v[i]))
                                lswitch = i;
                        else if(!safetopass(fwdangle, aftangle, angle[car],
                                lane[car], lane[i]))
                                   slowdown = 1;
                }
        }
}
        if((lswitch > -1) && (slowdown == 1))
        {
                oldspeed[car] = v[car];
                v[car] = v[lswitch];
                following[car] = lswitch;
                lswitch = 0;
        }
        else if(lswitch > -1)
        {
                lswitch = 1;
                lane[car  = 1 - lane[car];
        }
        else lswitch = 0;
        return(lswitch);
}

int needtopass(refer, angle, mylane, hislane, myspeed, hisspeed)
float refer, angle, myspeed, hisspeed;
int mylane, hislane;
{
        if((refer < (angle + .05)) && (refer > angle) &&
           (hislane == mylane) && (myspeed > hisspeed))
                return(1);
        else
                return(0);
}

int safetopass(fwd, aft, angle, mylane, hislane)
float fwd, aft, angle;
int mylane, hislane;
{
        if((fwd < (angle + .05)) && (aft > (angle - .02)) &&
           (mylane != hislane))
                return(0);
        else
                return(1);
```

```
}

refangles(fwd, aft, me, him)
float *fwd, *aft;
int me, him;
{
        if(((angle[me] + .05) >= 1.0) && (angle[him] < 0.05))
                *fwd = angle[him] + 1.0;
        else
                *fwd = angle[him];
        if(((angle[me] - .02) < 0.0) && (angle[him] > 0.98))
                *aft = 1.0 - angle[him];
        else
                *aft = angle[him];
}
```

```
/*****************************************************************************
        hostile.c

        Module that controls computer cars in "hostile cars" case.  Provides
        reactions to subject's attempts to pass.

*****************************************************************************/
#include "cars.h"
#include "carextvar.h"

stoppass(car, hostility)
int car, *hostility;
{
        int i;
        int lswitch = 0;
        extern float normal();
        static int timer[MYCAR] = {0, 0, 0, 0};

        if(*hostility == 2)
        {
                lswitch = varylane(car);
                if(following[car] > -1)
                        *hostility = 0;
        }
        if(*hostility < 2)
        {
        for(i=0; i<MYCAR; ++i)
        {
                if((i != car) && (lane[i] != lane[car]))
                {
                        if((fabs(angle[car] - angle[i]) < 0.02))
                        {
                                if(oldspeed[car] == 0.0)
                                        oldspeed[car] = v[car];
                                v[car] = v[i];
                                *hostility = 2;
                                timer[car] = 0;
                        }
                }
        }
        }
        if(*hostility == 0)
        {
                oldspeed[car] = v[car];
                timer[car] = 20;
                *hostility = 1;
        }
        if(*hostility == 1)
        {
                timer[car]--;
                v[car] = v[car] + .2;
                if(timer[car] <= 0)
                {
                        lswitch = 1;
```

```
                        lane[car] = 1 - lane[car];
                        *hostility = 0;
                        if(oldspeed[car] > 0.0)
                        {
                                v[car] = oldspeed[car];
                                oldspeed[car] = 0.0;
                        }
                        timer[car] = 0;
                }
        }
        return(lswitch);
}
```

```
/*****************************************************************************
        drive.c

        Module for controlling the subject's/net's car.
*****************************************************************************/
float cost[400], sint[400];
#include "cars.h"
#include "carextvar.h"

idrive()  /* determine the commanded speed and lane for the subject's car  */
{
        putmycar(BLACK);
        if(getvaluator(MOUSEX)<(CENTERX + 350))
                lane[MYCAR] = 0;
        else
                lane[MYCAR] = 1;
        putmycar(BLUE);
        v[MYCAR] = MAXVEL * (1 - (((CENTERY + 100) - getvaluator(MOUSEY)) /
                                  200.0));
}

putmycar(color)    /* place subject's car in the proper lane  */
int color;
{
        int x;

        x = CENTERX + (lane[MYCAR] * LANEWID + LANEDIAM);
        solidcirc(x, CENTERY, 8, color);
}


putcar(i, lswitch, score)  /* place computer car in new position  */
int i, lswitch, *score;
{
        int index, x, y;
        float z, w;

        index = angle[i] * 400.0;
        z = lane[i];
        if(lswitch == 0)
                w = lane[i];
        else
                w = 1 - lane[i];
        x = CENTERX + cost[index] * (w * LANEWID + LANEDIAM);
        y = CENTERY + sint[index] * (w * LANEWID + LANEDIAM);
        solidcirc(x, y, 8, BLACK);
        computeangles(i, score);
        index = angle[i] * 400.0;
        x = CENTERX + cost[index] * (z * LANEWID + LANEDIAM);
        y = CENTERY + sint[index] * (z * LANEWID + LANEDIAM);
        solidcirc(x, y, 8, RED);
}

putmarker()
```

```
{
        int index, x1, y1, x2, y2;
        static float marker;

        index = marker * 400.0;
        x1 = CENTERX + cost[index] * 161;
        y1 = CENTERY + sint[index] * 161;
        x2 = CENTERX + cost[index] * 199;
        y2 = CENTERY + sint[index] * 199;
        color(BLACK);
        move2i(x1, y1);
        draw2i(x2, y2);
        marker = marker - v[MYCAR] * DT;
        if(marker > 1.0) marker = marker - 1.0;
        if(marker < 0.0) marker = marker + 1.0;
        index = marker * 400.0;
        x1 = CENTERX + cost[index] * 161;
        y1 = CENTERY + sint[index] * 161;
        x2 = CENTERX + cost[index] * 199;
        y2 = CENTERY + sint[index] * 199;
        color(WHITE);
        move2i(x1, y1);
        draw2i(x2, y2);
}

int collision(i)
int i;
{
        if(((angle[i] > 0.9925) || (angle[i] < 0.0075)) &&
            (lane[i] == lane[MYCAR]))
                return(1);
        else
                return(0);
}
scoreboard(left, right, value, oldvalue, zero)
int left, right, oldvalue, value, zero;
{
        if(oldvalue > zero)
                solidbox(left, right, oldvalue, zero, BLACK);
        else
                solidbox(left, right, zero, oldvalue, BLACK);
        if(value > zero)
                solidbox(left, right, value, zero, RED);
        else
                solidbox(left, right, zero, value, RED);
}
```

```
/*****************************************************************************
        init.c

        Functions to initialize screens and variables.

****************************************************** ********************/
#include "cars.h"
#include "carextvar.h"

initscreen()
{
        int x, y;

        x = getvaluator(MOUSEX);
        y = getvaluator(MOUSEY);
        cursoff();
        ginit();
        color(BLACK);
        clear();
        setvaluator(MOUSEX, 0, 0, 1);
        setvaluator(MOUSEY, 0, 0, 1);
        circle(CENTERX, CENTERY, 200, WHITE);
        circle(CENTERX, CENTERY, 180, WHITE);
        circle(CENTERX, CENTERY, 160, WHITE);
        box((CENTERX + 300), (CENTERX + 350), (CENTERY - 100), (CENTERY + 100),
            WHITE);
        box((CENTERX + 350), (CENTERX + 400), (CENTERY - 100), (CENTERY + 100),
            WHITE);
        solidbox((CENTERX + 401), (CENTERX + 410), (CENTERY + 100),
                 (CENTERY - 100), YELLOW);
        setvaluator(MOUSEX, x, (CENTERX + 300), (CENTERX + 400));
        setvaluator(MOUSEY, y, (CENTERY - 100), (CENTERY + 100));
        color(WHITE);
        cmov2i((CENTERX + 387), (CENTERY + 110));
        charstr("TIME");
        cmov2i((CENTERX + 272), (CENTERY + 110));
        charstr("SCORE");
        curson();
}

unscore()
{
        color(BLACK);
        cmov2i((CENTERX + 272), (CENTERY + 110));
        charstr("SCORE");
        color(WHITE);
}

initvar()  /* set up initial values of several variables  */
{
        long timer;
        float x;
        int i, j, k;
        unsigned int l;
```

```
        extern float normal();

        l = time(&timer);
        srand48(timer);
        for (i = 0; i < 400; i++)
        {
                x = i / 400.0 * 2 * PI;
                cost[i] = cos(x);
                sint[i] = sin(x);
        }
        randangle();
        for(i=0; i<MYCAR; ++i)
        {
                v[i] = normal(8.0, 2.0);
        }
        v[MYCAR] = 8.0;
}

randangle()  /*  Set up initial random starting positions  */
{
        int i;

        for(i=0; i<MYCAR; ++i)
                angle[i] = drand48();
        angle[MYCAR] = 0.0;
}
```

```
/*****************************************************************************
        util.c

        Contains miscellaneous functions for the driving programs.

*****************************************************************************/
#include "cars.h"
#include "carextvar.h"

float normal(mean, sdev)
float mean, sdev;
{
int i;
float s = 0.0;

   for (i=0; i<12; ++i)
        s = s + drand48();
   s = (s - 6.0) * sdev + mean;
   return(s);
}

computeangles(i, score)
int i, *score;
{

      angle[i] = angle[i] + (v[i] - v[MYCAR]) * DT;
      if (angle[i] > 1.0)
         angle[i] = angle[i] - 1.0;
      if (angle[i] < 0.0)
      {
         angle[i] = angle[i] + 1.0;
         (*score)++;
      }
}

copy(s1, s2, start, lim)
char s1[], s2[];
int start, lim;
{
        int i;

        for(i=0; i<lim; ++i)
        {
                s2[i+start] = s1[i];
                if(s1[i] == '\0') break;
        }
        i = i + start;
        return(i);
}

length(c)
char *c;
{
        char *p = c;
```

```
        while (*p != '\0')
                p++;
        return(p - c);
}
```

```
/*****************************************************************************
        graph.c

        Graphics routines for the rest of the program.  Actual library function
        calls are Silicon Graphics specific, and would need to be replaced
        if this is implemented on another machine.

*****************************************************************************/

#include "gl.h"

circle(centerx, centery, radius, c)  /*  Draw a circle at the given x and y  */
                                     /*  center and radius in color c        */
int centerx, centery, radius, c;
{
        color(c);
        circi(centerx, centery, radius);
}

box(left, right, top, bottom, c)     /*  Draw a box between left and right,  */
                                     /*  top and bottom in color c           */
int left, right, top, bottom, c;
{
        color(c);
        recti(left, bottom, right, top);
}

solidcirc(centerx, centery, radius, c)/*  Draw a filled circle at the given  */
                                      /*  x and y center and radius  */
                                      /*  in color c        */
int centerx, centery, radius, c;
{
        color(c);
        circfi(centerx, centery, radius);
}
solidbox(left, right, top, bottom, c) /*  Draw a filled box between left and */
                                      /*  right, top and bottom in color c   */
int left, right, top, bottom, c;
{
        color(c);
        rectfi(left, top, right, bottom);
}

header(title, len, i)  /*  Put explanatory text on the screen   */
char *title;
int len, i;
{
        color(WHITE);
        cmov2i(327, 50);
        charstr("Press Left Mouse to Start Data Collection");
        cmov2i((513 - ((len * 9) / 2)), 700);
        charstr(title);
        cmov2i(490, 684);
        charstr("RUN ");
```

```
        if(i == 0)
                charstr("1");
        else
                charstr("2");
}

playhead(title, len)  /*  put title heading for playback programs   */
char *title;
int len;
{
        color(WHITE);
        cmov2i((513 - ((len * 9) / 2)), 700);
        charstr(title);
}

line2(a, b)   /*  put second explanatory line of text on screen for playback  */
int a, b;
{
        char line[50];
        int len;

        line[0] = 0;
        solidbox(200, 700, 675, 695, BLACK);
        color(WHITE);
        strcat(line, "Run Type ");
        len = strlen(line);
        line[len + 1] = 0;
        line[len] = b;
        if(a<0)
        {
                strcat(line, " Trained Net ");
        }
        else
        {
                strcat(line, "  Trial ");
                len = strlen(line);
                line[len + 1] = 0;
                line[len] = '0' + a;
        }
        cmov2i((513 - ((strlen(line) * 9) / 2)), 680);
        charstr(line);
}

footer()   /*  put bottom line telling whether practice of data run  */
{
        color(BLACK);
        cmov2i(327, 50);
        charstr("Press Left Mouse to Start Data Collection");
        color(WHITE);
        cmov2i(445, 50);
        charstr("Collecting Data");
}

zeroline(y, st_x, end_x)  /* draw a horizontal line from st_x to end_x at y  */
```

```
int y, st_x, end_x;
{
        movei(st_x, y, 0);
        color(WHITE);
        drawi(end_x, y, 0);
}
```

```
/*****************************************************************************
     cars.h

     Definitions and includes for the driving routines.
*****************************************************************************/


#include "stdio.h"
#include "math.h"
#include "time.h"
#include "gl.h"
#include "device.h"

#define PI 3.1415926
#define DT .001
#define NUMCARS 5
#define MYCAR 4
#define MAXVEL 15
#define DELAY 4000
#define CENTERX 325
#define CENTERY 400
#define LANEWID 20
#define LANEDIAM 170
#define END 180
```

```
/*****************************************************************************
        carvars.h

        Global variable declarations for the program module containing main()

*****************************************************************************/
float cost[400], sint[400];
float v[NUMCARS], angle[NUMCARS];
int lane[NUMCARS] = {0,0,1,1,0};
int following[NUMCARS] = {-1, -1, -1, -1, -1};
float oldspeed[MYCAR] = {0.0, 0.0, 0.0, 0.0};
float cost[400], sint[400];
```

```
/*****************************************************************************

     carextvar.h

     External global variable declarations for the driving routines
     external modules.
*****************************************************************************/

extern float cost[], sint[], v[], angle[], oldspeed[];
extern int lane[], following[];
```

Appendix C

Network C Source Code

```
/*****************************************************************************
*   File "vcontrn.c"            Trains net on 4-car continuous data.          *
*                               Variable Cars                                 *
*                               Input file is "*.sev".  Output file is        *
*   Revised:                    "*.wev" for weights, and "*.eev" for training *
*   29 Jul 89                   error data.                                   *
*                                                                             *
*                               Makes cvtrn.exe                               *
*                                                                             *
*                                Requires:                                    *
*                                     weights.c                               *
*                                     backprop.c                              *
*                                     net.c                                   *
*                                     inputs.c                                *
*                                     util.c                                  *
*                                                                             *
*                                                                             *
*                                                                             *
*****************************************************************************/
/*****************************************************************************
*   File "hcontrn.c"            Trains net on 4-car continuous data.          *
*                               Hostile scenario                              *
*                               Input file is "*.seh".  Output file is        *
*   Revised:                    "*.weh" for weights, and "*.eeh" for training *
*   29 Jul 89                   error data.                                   *
*                                                                             *
*                         Makes chtrn.exe                                     *
*                                                                             *
*                               Requires:                                     *
*                                     weights.c                               *
*                                     backprop.c                              *
*                                     net.c                                   *
*                                     inputs.c                                *
*                                     util.c                                  *
*                                                                             *
*****************************************************************************/
#include "cars.h"
#include "fourcars.h"
#include "net.h"
#include "fourin.h"
#include "carvars.h"
#include "netvars.h"

main(argc, argv)
int argc;
char *argv[];
{
        int i, j, k, count, attention, lan = 0;
        float outvec[2];
        float averror, oldv, oldl, eta, alpha, goodenuff, speed = 0.0;
        FILE *err, *in, *indat;
        long timer, l;
        char fname[50], errname[50], wtname[50], rawname[50], indatname[50];
        char *path, *trainext, *errext, *wtext, *rawwts;
```

```
  extern long time();
  extern FILE *fopen();
  extern float backprop();

  path = "hed$disk:[efix.nnets.driver.data]";
  if(argc == 0)
  {
printf("Useage: \"CVTRN subject_name [input_dat_file] [attention]\"\n");
          exit(0);
  }
  if(argc >= 3)
  {
          strcpy(indatname, path);
          strcat(indatname, argv[2]);
          indat = fopen(indatname, "r");
          if(indat == 0)
          {
                  printf("Input Data File %s not found\n", argv[2]);
                  exit(0);
          }
          fscanf(indat, "%f %f %f", &eta, &alpha,
                                        &goodenuff, &attention);
          fclose(indat);
  }
  else
  {
          eta = ETA;
          alpha = ALPHA;
          goodenuff = GOODENUFF;
          attention = 0;
  }
  printf("%d\n%s\n%s\n", argc, argv[1], argv[2]);
  printf("eta = %f alpha = %f goodenuff = %f, attention = %d\n",
                  eta, alpha, goodenuff, attention);
  trainext = ".sev";  /*  ".seh" for hostile cars   */
  errext = ".eev";    /*  ".eeh" for hostile cars   */
  wtext = ".wev";     /*  ".weh" for hostile cars   */
  rawwts = "rawwts.txt";
  strcpy(fname, path);
  strcat(fname, argv[1]);
  strcat(fname, trainext);
  strcpy(errname, path);
  strcat(errname, argv[1]);
  strcat(errname, errext);
  strcpy(wtname, path);
  strcat(wtname, argv[1]);
  strcat(wtname, wtext);
  strcpy(rawname, path);
  strcat(rawname, rawwts);
  initial(wtname, rawname, &count, I, &speed, &lan);
  for(i=0; i<10; ++i)
          get4inputs(oldv, oldl, I);  /*  Load Initial Inputs  */
  for(k=0; k<1000; ++k)
  {
```

```
for(j=0; j<J; ++j)
        hidden[j] = 0.0;
j = 0;
averror = 0.0;
if(k >= 5)
{
        attention = 1;
}
in = fopen(fname, "r");
if(in == 0)
{
        printf("%s not found.\n", fname);
        exit(0);
}
while(1)
{
        for(i=0; i<MYCAR; ++i)
        {
                fscanf(in, "%f %f %d", &v[i], &angle[i],
                                        &lane[i]);
                if((v[i] == 0.0) && (angle[i] == 0.0) &&
                   (lane[i] == 0))   /*  End of the data run  */
                        goto iterate;
        }
        oldv=v[MYCAR] / MAXVEL;
        oldl=lane[MYCAR];
        fscanf(in, "%f %d", &v[MYCAR], &lane[MYCAR]);
        outvec[SPEED] = v[MYCAR] / (MAXVEL + 5.0);
        outvec[LANE] = lane[MYCAR];
        get4inputs(oldv, oldl, I);
        if((input[I-6] < 0.2) && (input[I-5] < 0.2) &&
           (fabs(input[I-6] - input[I-5]) < 0.07) &&
                                attention)
        averror = averror + backprop(outvec, I, eta, alpha,
                                goodenuff, CLOSEIN, 1);
        else
        averror = averror + backprop(outvec, I, eta, alpha,
                                goodenuff, NORMAL, attention);
        count++;
        j++;
}
iterate:
        fclose(in);
        averror = averror / j;
        err = fopen(errname, "a");
        if(err == 0)
        {
                printf("%s not found.\n", fname);
                exit(0);
        }
        fprintf(err, "%5d, %f\n", count, averror);
        fclose(err);
        wtsave(wtname, count, I, speed, lan);
        if(averror < goodenuff)
```

```
                                goto end;
        }
        end:;
}
```

```
/*****************************************************************************
     backprop.c

     This module implements the back-propagation algorithm modified by
     Fix.

*****************************************************************************/
#include "net.h"
#include "netextvar.h"


float backprop(outvec, I, eta, alpha, goodenuff, version, attention)
     float outvec[], eta, alpha, goodenuff;
     int I, version, attention;
{
     float sum[J];
     float delta, cost, temp, flip;
     int i, j, k, l, v;
     extern float fabs();

  for(j=0; j<J; ++j) sum[j]=0;
       net(I, version);
     cost = (fabs(output[0] - outvec[0]) + fabs(output[1] - outvec[1])/
          (0.1 / goodenuff)) / K;
     for(k=0; k<K; ++k)
     {
          flip = sign(outvec[k] - 0.5) * (output[k] - 0.5);
               delta = (output[k] * (1 - output[k]) * u(flip) +
                0.25 * u(-flip)) * (outvec[k] - output[k]);
               for(l=0; l<J; ++l)
               {
               j = l + (J * version);
                    temp = w2[j][k] + eta * delta *
                    hidden[l] + alpha * (w2[j][k] - w2p[j][k]);
                  w2p[j][k] = w2[j][k];
                  w2[j][k] = temp;
                     sum[l] = sum[l] + delta * w2[j][k];
               if((!attention) && (MAXVERSIONS > 0))
               {
                    for(v=1; v<MAXVERSIONS; ++v)
                         w2[l+(v*J)][k] = w2[l][k];
               }
             }
             thetak[k + (K * version)] = thetak[k + (K * version)] -
                              eta * delta;
          if((!attention) && (MAXVERSIONS > 0))
          {
               for(v=1; v<MAXVERSIONS; ++v)
                    thetak[k+(v*K)] = thetak[k];
          }
     }
     for(j=0; j<J; ++j)
     {
          delta = hidden[j] * (1 - hidden[j]) * sum[j];
```

```
            for(l=0; l<I; ++l)
            {
                  i = l + (I * version);
                  temp = w1[i][j] + eta * delta * (input[l] - 0.5) +
                      alpha * (w1[i][j] - w1p[i][j]);
                  w1p[i][j] = w1[i][j];
                  w1[i][j] = temp;
                 if((!attention) && (MAXVERSIONS > 0))
                 {
                      for(v=1; v<MAXVERSIONS; ++v)
                           w1[l+(v*I)][j] = w1[l][j];
                 }
            }
            thetaj[j + (J * version)] = thetaj[j + (J * version)] -
                                        eta * delta;
            if((!attention) && (MAXVERSIONS > 0))
            {
                 for(v=1; v<MAXVERSIONS; ++v)
                      thetaj[j+(v*J)] = thetaj[j];
            }
            oldhidwts[j] = oldhidwts[j] + eta * delta * oldhidden[j];
      }
      return(cost);
}

int sign(x)     /*   signum function  */
float x;
{
      int i;

      if(x < 0)
            i = -1;
      else
            i = 1;

      return i;
}

int u(x)  /*  unit step function */
float x;
{
      int i;

      if(x < 0)
            i = 0;
      else
            i = 1;

      return i;
}
```

```
/*****************************************************************************
*                                                                            *
*    File "net.c"                        Calculates network output from      *
*                                        input array using weights matrices  *
*    Revised:                                                                *
*    30 Jul 89                           Uses Recursion                      *
*                                                                            *
*                                        Also uses redundant weights for     *
*                                        multiple conditions.                *
*                                                                            *
*                                                                            *
*****************************************************************************/
#include "net.h"
#include "netextvar.h"

net(I, version)
int I, version;
{
     extern float sigmoid();
     float x;
     int i, j, k;

for(j=0; j<J; ++j)
{
     x = 0.0;
     for(i=0; i<I; ++i)
     {
          x = x + w1[i + (I * version)][j] * (input[i] - 0.5);
     }
     x = x + oldhidwts[j] * hidden[j];
     oldhidden[j] = hidden[j];
     hidden[j] = sigmoid(x-thetaj[j + (J * version)]);
}

for(k=0; k<K; ++k)
{
     x = 0.0;
     for(j=0; j<J; ++j)
     {
          x = x + w2[j + (J * version)][k] * hidden[j];
     }
     output[k] = sigmoid(x-thetak[k + (K * version)]);
}
}

/*  SIGMOID FUNCTION         */

float sigmoid(x)
     float x;
{    float y;
     extern float exp();

     if((BETA * x) < -50)
        y = 0.0;
```

```
        else if((BETA * x) > 50)
           y = 1.0;
        else
           y = 1 / (1 + exp(-BETA * x));
        return (y);
}
```

```
/*****************************************************************************
        weights.c

        Module contains routines to save and retrieve weights from
        files.

*****************************************************************************/
#include "cars.h"
#include "net.h"
#include "carextvar.h"
#include "netextvar.h"

wtsave(fname, count, I, speed, lane)
     char fname[];
     int count, I, lane;
     float speed;
{    extern FILE *fopen();
     FILE *dat;
     int i, j, k, l;

     remove(fname);
     dat = fopen(fname, "w");
     if(dat == 0) exit(0);

     for(j=0; j<(J*MAXVERSIONS); ++j)
     {
             fprintf(dat, "%13.10f ", thetaj[j]);
     }
     fprintf(dat, "\n");

     for(k=0; k<(K*MAXVERSIONS); ++k)
     {
             fprintf(dat, "%13.10f ", thetak[k]);
     }
     fprintf(dat, "\n");

     for(j=0; j<J; ++j)
     {
             fprintf(dat, "%13.10f ", oldhidwts[j]);
     }
     fprintf(dat, "\n");

     for(i=0; i<(I*MAXVERSIONS); ++i)
     {
          for(j=0; j<J; ++j)
          {
               fprintf(dat, "%13.10f ", wi[i][j]);
          }
          fprintf(dat, "\n");
     }

     for(j=0; j<(J*MAXVERSIONS); ++j)
     {
          for(k=0; k<K; ++k)
```

```
          {
               fprintf(dat, "%13.10f ", w2[j][k]);
          }
          fprintf(dat, "\n");
     }
     fprintf(dat,"%d %6.4f %d\n", count, speed, lane);
     fclose(dat);
}

initial(fname, rawname, count, I, speed, lane)
     char fname[], rawname[];
     int *count, I, *lane;
     float *speed;
{    extern FILE *fopen();
     FILE *dat;
     int i, j, k, new_flag = 0;

     dat = fopen(fname, "r");
     if(dat == 0)
     {
        printf("No previous weights file\n");
        new_flag = 1;
        dat = fopen(rawname, "r");
        if(dat == 0)
        {
                printf("No raw weights file\n");
                exit(1);
        }
     }
     for(j=0; j<(J*MAXVERSIONS); ++j)
{
          fscanf(dat, "%f", thetaj+j);
}

     for(k=0; k<(K*MAXVERSIONS); ++k)
{
            fscanf(dat, "%f", thetak+k);
}

     for(j=0; j<J; ++j)
        {
             fscanf(dat, "%f ", &oldhidwts[j]);
        }

     for(i=0; i<(I*MAXVERSIONS); ++i)
     {
          for(j=0; j<J; ++j)
          {
               fscanf(dat, "%f ", &w1[i][j]);
               w1p[i][j] = w1[i][j];
          }
     }

     for(j=0; j<(J*MAXVERSIONS); ++j)
```

```
    {
        for(k=0; k<K; ++k)
        {
            fscanf(dat, "%f ", &w2[j][k]);
            w2p[j][k] = w2[j][k];
        }
    }


    if(!new_flag)
        fscanf(dat, "%d %f %d", count, speed, lane);
    else
    {
       *count = 0;
       *speed = 0.0;
       *lane = 0;
       for(j=0; j<J; ++j)
            oldhidwts[j] = 0.0;
    }

    fclose(dat);
}
```

```
/*****************************************************************************
*                                                                            *
*    File "inputs.c"                 Sets up inputs for the nets.            *
*                                    "get4inputs()" sets up inputs for the   *
*    Revised:                        case of 4 cars and continuous data, and *
*    20 Jun 89                       "get2inputs()" sets up input vectors for*
*                                    the case of multiple setups and two     *
*                                    other cars.                             *
*                                                                            *
*                                                                            *
*                                                                            *
*****************************************************************************/
#include "net.h"
#include "cars.h"
#include "netextvar.h"
#include "carextvar.h"

get4inputs(s, l, I)
int I;
float s, l;
{
        int i, j, k;
        int lane0[4] = {VOIDCAR, VOIDCAR, VOIDCAR, VOIDCAR};
        int lane1[4] = {VOIDCAR, VOIDCAR, VOIDCAR, VOIDCAR};
        int count0 = 0;
        int count1 = 0;

        for(i=0; i<4; ++i)
        {
                if(lane[i] == 0)
                        lane0[count0++] = i;
                else
                        lane1[count1++] = i;
        }
        j = 1;
        while(j==1)
        {
                j = 0;
                for(i=0; i<3; ++i)
                {
                        if((lane0[i] != VOIDCAR) && (lane0[i+1] != VOIDCAR))
                        {
                        if(angle[lane0[i]] > angle[lane0[i+1]])
                                j = swap(&lane0[i], &lane0[i+1]);
                        }
                        if((lane1[i] != VOIDCAR) && (lane1[i+1] != VOIDCAR))
                        {
                        if(angle[lane1[i]] > angle[lane1[i+1]])
                                j = swap(&lane1[i], &lane1[i+1]);
                        }
                }
        }
        for(i=0; i<I-6; ++i)
                input[i] = input[i+4];
```

```
        for(k=0; k<2; ++k)
        {
                i = k * 2 + (I - 6);
                if(lane0[k] != VOIDCAR)
                {
                        input[i] = angle[lane0[k]];
                }
                else
                {
                        input[i] = 0.75;
                }
                if(lane1[k] != VOIDCAR)
                {
                        input[i+1] = angle[lane1[k]];
                }
                else
                {
                        input[i+1] = 0.75;
                }
        }
        input[I-2] = s;
        input[I-1] = l;
}
```

```
/*****************************************************************************
        util.c

        Module contains miscellaneous functions for the driving routines.

*****************************************************************************/

#include "cars.h"
#include "carextvar.h"

float normal(mean, sdev)
float mean, sdev;
{
        int i;
        float s = 0.0;
        extern float frandom();

           for (i=0; i<12; ++i)
                s = s + frandom();
           s = (s - 6.0) * sdev + mean;
           return(s);
}

computeangles(i, score, MYCAR)
int i, *score, MYCAR;
{

      angle[i] = angle[i] + (v[i] - v[MYCAR]) * DT;
      if (angle[i] > 1.0)
         angle[i] = angle[i] - 1.0;
      if (angle[i] < 0.0)
      {
         angle[i] = angle[i] + 1.0;
         (*score)++;
      }
}

int swap(a, b)
int *a, *b;
{
        int c;

        c = *a;
        *a = *b;
        *b = c;

        return(1);
}

float frandom()
{
        float x;

        x = (float) rand() / 2147483646.0;
```

```
        return(x);
}

int collision(i, MYCAR)
int i, MYCAR;
{
        if(((angle[i] > 0.9925) || (angle[i] < 0.0075)) &&
            (lane[i] == lane[MYCAR]))
                return(1);
        else
                return(0);
}
```

```
/*****************************************************************************

                                net.h

                          Defines for network

*****************************************************************************/
#define J 20    /*  Number of Hidden Nodes  /*
#define K 2     /*  Number of outputs       /*
#define GOODENUFF .01
#define ETA 0.7
#define ALPHA 0.3
#define BETA 1.0
#define LANE 1
#define SPEED 0
#define MAXVERSIONS 2
#define NORMAL 0
#define CLOSEIN 1
```

```
/*****************************************************************************

                                fourin.h

          Sets network input size for four cars, continuous data

*****************************************************************************/
#define I 42
```

```
/*****************************************************************************

                                    car.h

                Defines and Includes for the car programs

*****************************************************************************/
#include <stdio.h>
#include <math.h>
#include <time.h>

#define PI 3.1415926
#define DT .001
#define VOIDCAR -1
#define MAXVEL 15
#define DELAY 4000
#define CENTERX 325
#define CENTERY 400
#define LANEWID 20
#define LANEDIAM 170
#define END 180
#define TYPES 14
#define TIME_UP 8535
```

```
/*****************************************************************************

                                carvars.h

                        Global car variable declarations

*****************************************************************************/
float v[NUMCARS], angle[NUMCARS];
int lane[5] = {0,0,1,1,0};
int following[5] = {-1, -1, -1, -1, -1};
float oldspeed[4] = {0.0, 0.0, 0.0, 0.0};
```

```
/*****************************************************************************

                              fourcars.h

              Defines of number of cars for continuous data

*****************************************************************************/
#define NUMCARS 5
#define MYCAR 4
```

```
/*****************************************************************************

                              netvars.h

                    Global variables for the network

*****************************************************************************/
float input[I], hidden[J], oldhidden[J], output[K];
float w1[(I*MAXVERSIONS)][J], w2[(J*MAXVERSIONS)][K],
        w1p[(I*MAXVERSIONS)][J], w2p[(J*MAXVERSIONS)][K];
float thetak[(K*MAXVERSIONS)], thetaj[(J*MAXVERSIONS)], oldhidwts[J];
```

```
/*****************************************************************************

                                netextvar.h

          External global variable declaration for the network.

*****************************************************************************/
extern float input[], hidden[], oldhidden[], output[], w1[][J], w2[][K],
             w1p[][J], w2p[][K], thetak[], thetaj[], oldhidwts[J];
```

Appendix D

Net Testing C Source Code

```
/*****************************************************************************
*   File "vcontest.c"              Tests trained net on continuous data      *
*                                  with four other cars                      *
*   Revised:                                                                 *
*    5 Sep 89                                                                *
*                                  Makes cvtest.exe                          *
*                                                                            *
*                                  Requires:                                 *
*                                     weights.c                              *
*                                     net.c                                  *
*                                     inputs.c                               *
*                                     drive.c                                *
*                                     util.c                                 *
*                                     initrand.c                             *
*                                                                            *
*****************************************************************************/
#include "cars.h"
#include "fourcars.h"
#include "net.h"
#include "fourin.h"
#include "carvars.h"
#include "netvars.h"

main(argc, argv)
int argc;
char *argv[];
{

        int i, j, k, lswitch, score, testnr, testype, flag1, flag2, lan = 0;
        float speed = 0.0;
        static int bump[MYCAR] = {0, 0, 0, 0};
        FILE *out, *testin;
        long endtime, timer;
        char outname[50], wtname[50], fname[50];
        char *path, *wtext, *outext;
        extern FILE *fopen();

        fname[0] = 0;
        outname[0] = 0;
        wtname[0] = 0;
        path = "hed$disk:[efix.nnets.driver.data]";
        wtext = ".wev";
        outext = ".nev";
        strcat(fname, path);
        strcat(outname, path);
        strcat(outname, argv[1]);
        strcat(outname, outext);
        strcat(wtname, path);
        strcat(wtname, argv[1]);
        strcat(wtname, wtext);
        initial(wtname, wtname, &k, I, &speed, &lan);
        out = fopen(outname, "w");
        if(out == 0) exit(1);
        seed();
```

```
        randangle(MYCAR);
        randvel(MYCAR);
        for(i=0; i<MYCAR; ++i)
                computeangles(i, &j, MYCAR);
        for(j=0; j<J; ++j)
                hidden[j] = 0.0;
        for (i=0; i<10; ++i)
                get4inputs((8.0/MAXVEL), 0.0, I);
        for(j=0; j<TIME_UP; ++j)
        {
                for(i=0; i<MYCAR; ++i)
                {
                        varyspeed(i);
                        lswitch = varylane(i, NUMCARS);
                        computeangles(i, &k, MYCAR);
                }
                if((input[I-5] < 0.2) && (input[I-6] < 0.2)
                   && (fabs(input[I-5] - input[I-6]) < 0.07))
                        netdrive(I, NUMCARS, MYCAR, CLOSEIN);
                else
                        netdrive(I, NUMCARS, MYCAR, NORMAL);
                for(i=0; i<MYCAR; ++i)
                {
                        fprintf(out, "%f ", v[i]);
                        fprintf(out, "%f %d ", angle[i], lane[i]);
                }
                fprintf(out, "%f %d\n", v[MYCAR], lane[MYCAR]);
        }
        fprintf(out, "%f %f %d", 0.0, 0.0, 0);
        fclose(out);
}
```

```
/*****************************************************************************
*  File "hcontest.c"              Tests trained net on continuous data,      *
*                                  four other cars, hostile scenario         *
*    Revised:                                                                *
*   18 May 89                     Makes the program chtest.exe              *
*                                                                           *
*                                 Requires:                                 *
*                                     weights.c                             *
*                                     net.c                                 *
*                                     inputs.c                              *
*                                     drive.c                               *
*                                     util.c                                *
*                                     initrand.c                            *
*                                     hostile.c                             *
*                                                                           *
*****************************************************************************/
#include "cars.h"
#include "fourcars.h"
#include "net.h"
#include "fourin.h"
#include "carvars.h"
#include "netvars.h"

main(argc, argv)
int argc;
char *argv[];
{
        int i, j, k, lswitch, score, testnr, testype, lan = 0;
        float speed = 0.0;
        static int bump[MYCAR] = {0, 0, 0, 0};
        static int hostility[MYCAR] = {0, 0, 0, 0};
        FILE *out, *testin;
        char fname[50], wtname[50], outname[50];
        char *path, *wtext, *outext;
        extern FILE *fopen();

        outname[0] = 0;
        wtname[0] = 0;
        fname[0] = 0;
        path = "hed$disk:[efix.nnets.driver.data]";
        wtext = ".weh";
        outext = ".neh";
        strcat(fname, path);
        strcat(outname, path);
        strcat(outname, argv[1]);
        strcat(outname, outext);
        strcat(wtname, path);
        strcat(wtname, argv[1]);
        strcat(wtname, wtext);
        initial(wtname, wtname, &k, I, &speed, &lan);
        out = fopen(outname, "w");
        if(out == 0) exit(1);
        seed();
        randangle(MYCAR);
```

```
        randvel(MYCAR);
        lane[MYCAR] = 0;
        for(i=0; i<MYCAR; ++i)
                computeangles(i, &j, MYCAR);
        for(i=0; i<10; ++i)
                get4inputs(8.0, 0.0, I);
        for(j=0; j<J; ++j)
                hidden[j] = 0.0;
        for(j=0; j<TIME_UP; ++j)
        {
                for(i=0; i<MYCAR; ++i)
                {
                        if((angle[i] < 0.1) && (lane[i] != lane[MYCAR]))
                        {
                                lswitch = stoppass(i, &hostility[i],
                                                MYCAR, NUMCARS);
                                computeangles(i, &k, MYCAR);
                        }
                        else
                        {
                                varyspeed(i);
                                lswitch =     lane(i, NUMCARS);
                                hostility[i] = 0;
                                if((following[i] == -1) && (oldspeed[i] > 0.0))
                                {
                                        v[i] = oldspeed[i];
                                        oldspeed[i] = 0.0;
                                }
                                computeangles(i, &k, MYCAR);
                        }
                }
                if((input[I-6] < 0.2) && (input[I-5] < 0.2) &&
                        (fabs(input[I-6] - input[I-5]) < 0.1))
                        netdrive(I, NUMCARS, MYCAR, CLOSEIN);
                else
                        netdrive(I, NUMCARS, MYCAR, NORMAL);
                for(i=0; i<MYCAR; ++i)
                {
                        fprintf(out, "%f ", v[i]);
                        fprintf(out, "%f %d ", angle[i], lane[i]);
                }
                fprintf(out, "%f %d\n", v[MYCAR], lane[MYCAR]);
        }
        fprintf(out, "%f %f %d\n", 0.0, 0.0, 0);
        fclose(out);
}
```

```
/*****************************************************************************
*                                                                            *
*    File "net.c"                        Calculates network output from      *
*                                        input array using weights matrices  *
*    Revised:                                                                *
*    30 Jul 89                           Uses Recursion                      *
*                                                                            *
*                                        Also uses redundant weights for     *
*                                        multiple conditions.                *
*                                                                            *
*                                                                            *
*****************************************************************************/
#include "net.h"
#include "netextvar.h"

net(I, version)
int I, version;
{
     extern float sigmoid();
     float x;
     int i, j, k;

for(j=0; j<J; ++j)
{
     x = 0.0;
     for(i=0; i<I; ++i)
     {
          x = x + w1[i + (I * version)][j] * (input[i] - 0.5);
     }
     x = x + oldhidwts[j] * hidden[j];
     oldhidden[j] = hidden[j];
     hidden[j] = sigmoid(x-thetaj[j + (J * version)]);
}

for(k=0; k<K; ++k)
{
     x = 0.0;
     for(j=0; j<J; ++j)
     {
          x = x + w2[j + (J * version)][k] * hidden[j];
     }
     output[k] = sigmoid(x-thetak[k + (K * version)]);
}
}

/*  SIGMOID FUNCTION         */

float sigmoid(x)
     float x;
{    float y;
     extern float exp();

     if((BETA * x) < -50)
        y = 0.0;
```

```
    else if((BETA * x) > 50)
       y = 1.0;
    else
       y = 1 / (1 + exp(-BETA * x));
    return (y);
}
```

```
/*****************************************************************************
*                                                                            *
*    File "inputs.c"                 Sets up inputs for the nets.            *
*                                    "get4inputs()" sets up inputs for the   *
*    Revised:                        case of 4 cars and continuous data, and *
*    20 Jun 89                       "get2inputs()" sets up input vectors for*
*                                    the case of multiple setups and two     *
*                                    other cars.                             *
*                                                                            *
*                                                                            *
*                                                                            *
*****************************************************************************/
#include "net.h"
#include "cars.h"
#include "netextvar.h"
#include "carextvar.h"

get4inputs(s, l, I)
int I;
float s, l;
{
        int i, j, k;
        int lane0[4] = {VOIDCAR, VOIDCAR, VOIDCAR, VOIDCAR};
        int lane1[4] = {VOIDCAR, VOIDCAR, VOIDCAR, VOIDCAR};
        int count0 = 0;
        int count1 = 0;

        for(i=0; i<4; ++i)
        {
                if(lane[i] == 0)
                        lane0[count0++] = i;
                else
                        lane1[count1++] = i;
        }
        j = 1;
        while(j==1)
        {
                j = 0;
                for(i=0; i<3; ++i)
                {
                        if((lane0[i] != VOIDCAR) && (lane0[i+1] != VOIDCAR))
                        {
                        if(angle[lane0[i]] > angle[lane0[i+1]])
                                j = swap(&lane0[i], &lane0[i+1]);
                        }
                        if((lane1[i] != VOIDCAR) && (lane1[i+1] != VOIDCAR))
                        {
                        if(angle[lane1[i]] > angle[lane1[i+1]])
                                j = swap(&lane1[i], &lane1[i+1]);
                        }
                }
        }
        for(i=0; i<I-6; ++i)
                input[i] = input[i+4];
```

```
        for(k=0; k<2; ++k)
        {
                i = k * 2 + (I - 6);
                if(lane0[k] != VOIDCAR)
                {
                        input[i] = angle[lane0[k]];
                }
                else
                {
                        input[i] = 0.75;
                }
                if(lane1[k] != VOIDCAR)
                {
                        input[i+1] = angle[lane1[k]];
                }
                else
                {
                        input[i+1] = 0.75;
                }
        }
        input[I-2] = s;
        input[I-1] = l;
}
```

```
/*****************************************************************************
        hostile.c

        Module with the logic for the adversary cars to work against
        the subject or net in the hostile cars scenario

*****************************************************************************/
#include "carextvar.h"

stoppass(car, hostility, MYCAR, NUMCARS)
int car, *hostility, MYCAR, NUMCARS;
{
        int i;
        int lswitch = 0;
        static int timer[4] = {0, 0, 0, 0};
        extern float normal();
        extern float fabs();

        if(*hostility == 2)
        {
                lswitch = varylane(car, NUMCARS);
                if(following[car] > -1)
                        *hostility = 0;
        }
        if(*hostility < 2)
        {
        for(i=0; i<MYCAR; ++i)
        {
                if((i != car) && (lane[i] != lane[car]))
                {
                        if((fabs(angle[car] - angle[i]) < 0.02))
                        {
                                if(oldspeed[car] == 0.0)
                                        oldspeed[car] = v[car];
                                v[car] = v[i];
                                *hostility = 2;
                                timer[car] = 0;
                        }
                }
        }
        }
        if(*hostility == 0)
        {
                oldspeed[car] = v[car];
                timer[car] = 20;
                *hostility = 1;
        }
        if(*hostility == 1)
        {
                timer[car]--;
                v[car] = v[car] + .2;
                if(timer[car] <= 0)
                {
                        lswitch = 1;
```

```
                        lane[car] = 1 - lane[car];
                        *hostility = 0;
                        if(oldspeed[car] > 0.0)
                        {
                                v[car] = oldspeed[car];
                                oldspeed[car] = 0.0;
                        }
                        timer[car] = 0;
                }
        }
        return(lswitch);
}
```

```
/*****************************************************************************
        initrand.c

        Module contains functions to initialize random scenarios and
        inject random variation into the scenarios.

*****************************************************************************/
#include "cars.h"
#include "carextvar.h"

randangle(MYCAR)   /*  initialize cars at random positions  */
int MYCAR;
{
        int i;
        extern float frandom();

        for(i=0; i<MYCAR; ++i)
        {
                angle[i] = frandom();
        }
        angle[MYCAR] = 0.0;
}

randvel(MYCAR)   /*  initialize cars at random speeds  */
int MYCAR;
{
        int i;
        extern float normal();

        for(i=0; i<MYCAR; ++i)
                v[i] = normal(8.0, 2.0);

        v[MYCAR] = 8.0;
}

varyspeed(car)
int car;
{
        float x;
        extern float frandom();
        extern float normal();

        x = frandom();
        if(x > .98)
                v[car] = normal(v[car], 2.0);
        if(v[car] > 12.0)v[car] = 12.0;
        if(v[car] < 3.0)v[car] = 3.0;
}

int varylane(car, NUMCARS)
int car, NUMCARS;
{
        int i;
        int lswitch = -1;
```

```
        int slowdown = 0;
        float fwdangle, aftangle;

if((following[car] > -1) && (lane[following[car]] != lane[car]))
{
        following[car] = -1;
        v[car] = oldspeed[car];
        oldspeed[car] = 0.0;
}
if(following[car] > -1)
{
        v[car] = v[following[car]];
        for(i=0; i<NUMCARS; ++i)
        {
                if((i != car) && (i != following[car]))
                {
                        refangles(&fwdangle, &aftangle, car, i);
                        if(safetopass(fwdangle, aftangle, angle[car],
                                        lane[car], lane[i]) == 0)
                                slowdown = 1;
                }
        }
        if(slowdown == 0)
        {
                v[car] = oldspeed[car];
                oldspeed[car] = 0.0;
                lswitch = 1;
                following[car] = -1;
        }
}
else
{
        for(i=0; i<NUMCARS; ++i)
        {
                if(i != car)
                {
                        refangles(&fwdangle, &aftangle, car, i);
                        if(needtopass(fwdangle, angle[car], lane[car], lane[i],
                                        v[car], v[i]))
                                lswitch = i;
                        else if(!safetopass(fwdangle, aftangle, angle[car],
                                lane[car], lane[i]))
                                   slowdown = 1;
                }
        }
}
        if((lswitch > -1) && (slowdown == 1))
        {
                oldspeed[car] = v[car];
                v[car] = v[lswitch];
                following[car] = lswitch;
                lswitch = 0;
        }
        else if(lswitch > -1)
```

```
        {
                lswitch = 1;
                lane[car] = 1 - lane[car];
        }
        else lswitch = 0;
        return(lswitch);
}

int needtopass(refer, angle, mylane, hislane, myspeed, hisspeed)
float refer, angle, myspeed, hisspeed;
int mylane, hislane;
{
        if((refer < (angle + .05)) && (refer > angle) &&
           (hislane == mylane) && (myspeed > hisspeed))
                return(1);
        else
                return(0);
}

int safetopass(fwd, aft, angle, mylane, hislane)
float fwd, aft, angle;
int mylane, hislane;
{
        if((fwd < (angle + .05)) && (aft > (angle - .02)) &&
           (mylane != hislane))
                return(0);
        else
                return(1);
}

refangles(fwd, aft, me, him)
float *fwd, *aft;
int me, him;
{
        if(((angle[me] + .05) >= 1.0) && (angle[him] < 0.05))
                *fwd = angle[him] + 1.0;
        else
                *fwd = angle[him];
        if(((angle[me] - .02) < 0.0) && (angle[him] > 0.98))
                *aft = 1.0 - angle[him];
        else
                *aft = angle[him];
}

seed()
{
        long timer, l;
        int seed;

        l = time(&timer);
        seed = (int)l;
        srand(seed);
}
```

```
/*****************************************************************************
        util.c

        Module contains miscellaneous functions for the driving routines.

*****************************************************************************/

#include "cars.h"
#include "carextvar.h"

float normal(mean, sdev)
float mean, sdev;
{
        int i;
        float s = 0.0;
        extern float frandom();

           for (i=0; i<12; ++i)
                s = s + frandom();
           s = (s - 6.0) * sdev + mean;
           return(s);
}

computeangles(i, score, MYCAR)
int i, *score, MYCAR;
{

      angle[i] = angle[i] + (v[i] - v[MYCAR]) * DT;
      if (angle[i] > 1.0)
         angle[i] = angle[i] - 1.0;
      if (angle[i] < 0.0)
      {
         angle[i] = angle[i] + 1.0;
         (*score)++;
      }
}

int swap(a, b)
int *a, *b;
{
        int c;

        c = *a;
        *a = *b;
        *b = c;

        return(1);
}

float frandom()
{
        float x;

        x = (float) rand() / 2147483646.0;
```

```
        return(x);
}

int collision(i, MYCAR)
int i, MYCAR;
{
        if(((angle[i] > 0.9925) || (angle[i] < 0.0075)) &&
            (lane[i] == lane[MYCAR]))
                return(1);
        else
                return(0);
}
```

```
/*****************************************************************************
        drive.c

        Module with netdrive() function to allow the network model to control
        the car.

*****************************************************************************/
#include "net.h"
#include "cars.h"
#include "carextvar.h"
#include "netextvar.h"

int netdrive(I, NUMCARS, MYCAR, versicn)
int I, NUMCARS, MYCAR, version;
{
        int i;
        float p;

        if(NUMCARS == 3)   /* then we're in the setups scenario with 2
                              adversary cars          */
                get2inputs((v[MYCAR] / MAXVEL), (float)lane[MYCAR], I);
        else if(NUMCARS == 5) /* then we're in the continuous data scenario
                                 with 4 adversary cars    */
                get4inputs((v[MYCAR] / MAXVEL), (float)lane[MYCAR], I);
        if(output[LANE] != 1.0 && output[LANE] != 0.0)
                output[LANE] = 0.0;
        p = output[LANE];
        net(I, version);
        if(fabs(output[LANE] - p) > 0.55)
                output[LANE] = 1.0 - p;
        else output[LANE] = p;
        lane[MYCAR] = (int)output[1];
        v[MYCAR] = (MAXVEL + 5.0) * output[SPEED];
        if(v[MYCAR] > MAXVEL) v[MYCAR] = MAXVEL;
        return(version);
}
```

```
/*****************************************************************************
        weights.c

        Module contains routines to save and retrieve weights from
        files.

*****************************************************************************/
#include "cars.h"
#include "net.h"
#include "carextvar.h"
#include "netextvar.h"

wtsave(fname, count, I, speed, lane)
     char fname[];
     int count, I, lane;
     float speed;
{    extern FILE *fopen();
     FILE *dat;
     int i, j, k, l;

     remove(fname);
     dat = fopen(fname, "w");
     if(dat == 0) exit(0);

     for(j=0; j<(J*MAXVERSIONS); ++j)
     {
             fprintf(dat, "%13.10f ", thetaj[j]);
     }
     fprintf(dat, "\n");

     for(k=0; k<(K*MAXVERSIONS); ++k)
     {
             fprintf(dat, "%13.10f ", thetak[k]);
     }
     fprintf(dat, "\n");

     for(j=0; j<J; ++j)
     {
             fprintf(dat, "%13.10f ", oldhidwts[j]);
     }
     fprintf(dat, "\n");

     for(i=0; i<(I*MAXVERSIONS); ++i)
     {
          for(j=0; j<J; ++j)
          {
               fprintf(dat, "%13.10f ", w1[i][j]);
          }
          fprintf(dat, "\n");
     }

     for(j=0; j<(J*MAXVERSIONS); ++j)
     {
          for(k=0; k<K; ++k)
```

```
          {
               fprintf(dat, "%13.10f ", w2[j][k]);
          }
          fprintf(dat, "\n");
     }
     fprintf(dat,"%d %6.4f %d\n", count, speed, lane);
     fclose(dat);
}

initial(fname, rawname, count, I, speed, lane)
     char fname[], rawname[];
     int *count, I, *lane;
     float *speed;
{    extern FILE *fopen();
     FILE *dat;
     int i, j, k, new_flag = 0;

     dat = fopen(fname, "r");
     if(dat == 0)
     {
        printf("No previous weights file\n");
        new_flag = 1;
        dat = fopen(rawname, "r");
        if(dat == 0)
        {
                printf("No raw weights file\n");
                exit(1);
        }
     }
     for(j=0; j<(J*MAXVERSIONS); ++j)
{
          fscanf(dat, "%f", thetaj+j);
}

     for(k=0; k<(K*MAXVERSIONS); ++k)
{
            fscanf(dat, "%f", thetak+k);
}

     for(j=0; j<J; ++j)
        {
             fscanf(dat, "%f ", &oldhidwts[j]);
        }

     for(i=0; i<(I*MAXVERSIONS); ++i)
     {
          for(j=0; j<J; ++j)
          {
               fscanf(dat, "%f ", &w1[i][j]);
               w1p[i][j] = w1[i][j];
          }
     }

     for(j=0; j<(J*MAXVERSIONS); ++j)
```

```
    {
        for(k=0; k<K; ++k)
        {
            fscanf(dat, "%f ", &w2[j][k]);
            w2p[j][k] = w2[j][k];
        }
    }


    if(!new_flag)
            fscanf(dat, "%d %f %d", count, speed, lane);
    else
    {
       *count = 0;
       *speed = 0.0;
       *lane = 0;
       for(j=0; j<J; ++j)
               oldhidwts[j] = 0.0;
    }

    fclose(dat);
}
```

```
/*****************************************************************************

                                net.h

                          Defines for network

*****************************************************************************/
#define J 20    /*  Number of Hidden Nodes  /*
#define K 2     /*  Number of outputs       /*
#define GOODENUFF .01
#define ETA 0.7
#define ALPHA 0.3
#define BETA 1.0
#define LANE 1
#define SPEED 0
#define MAXVERSIONS 2
#define NORMAL 0
#define CLOSEIN 1
```

```
/*****************************************************************************

                              fourin.h

          Sets network input size for four cars, continuous data

*****************************************************************************/
#define I 42
```

```
/*****************************************************************************

                              fourcars.h

                Defines of number of cars for continuous data

*****************************************************************************/
#define NUMCARS 5
#define MYCAR 4
```

```
/*****************************************************************************

                                    car.h

                Defines and Includes for the car programs

*****************************************************************************/
#include <stdio.h>
#include <math.h>
#include <time.h>

#define PI 3.1415926
#define DT .001
#define VOIDCAR -1
#define MAXVEL 15
#define DELAY 4000
#define CENTERX 325
#define CENTERY 400
#define LANEWID 20
#define LANEDIAM 170
#define END 180
#define TYPES 14
#define TIME_UP 8535
```

```
/*****************************************************************************

                                carvars.h

                        Global car variable declarations

*****************************************************************************/
float v[NUMCARS], angle[NUMCARS];
int lane[5] = {0,0,1,1,0};
int following[5] = {-1, -1, -1, -1, -1};
float oldspeed[4] = {0.0, 0.0, 0.0, 0.0};
```

```
/*****************************************************************************

                                netextvar.h

            External global variable declaration for the network.

*****************************************************************************/
extern float input[], hidden[], oldhidden[], output[], w1[][J], w2[][K],
             w1p[][J], w2p[][K], thetak[], thetaj[], oldhidwts[J];
```

```
/*****************************************************************************

                              carextvar.h

                    External Global Variable Declarations

*****************************************************************************/
extern int lane[], following[];
extern float v[], angle[], oldspeed[];
```

References

Almeida, Luis B., "A Learning Rule for Asynchronous Perceptrons with Feedback in a Combinatorial Environment," *Proc IEEE First Annual International Conference on Neural Networks*, 1:609–618, 1987

Fix, E.L., "Optimizing Back Propagation", *Proceedings, Aerospace Applications of Artificial Intelligence Conference*, 2:52–56, 1988.

Holden, Alistair D.C., "Fast Learning in Symbolic/Neural Models using External Constraints and Automatic Re-Structuring," *Proc IEEE Systems, Man, and Cybernetics*, 1:2–4, 1989.

Lippmann, R.P., "An Introduction to Neural Nets," *IEEE ASSP Magazine*, April, 1987, pp 4–22.

Pineda, Fernando J., "Generalization of Back-Propagation to Recurrent Neural Networks," Physical Review Letters, 59 (19), 2229, 1987

Restrepo, Pedro, Personal communication, Alphatech, Inc., 1988

Rumelhart, D.E., G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation," *Parallel Distributed Processing Volume 1: Foundations*, pp. 318–362, D. E. Rumelhart and J.L. McClelland (Eds.). Cambridge, MA: MIT Press/Bradford, 1986.

Shepanski, J.F., and S.A. Macy, "Manual Training of Autonomous Systems Based on Artificial Neural Networks," *Proceedings, IEEE International Conference on Neural Networks*, 4:697–704, 1987.

☆ U.S.Government Printing Office:1990—548-018/20211